A Review on Agent-to-Agent Protocol: Concept, State-of-the-art, Challenges and Future Directions

Partha Pratim Ray

***Abstract*—The evolution of autonomous multi-agent systems has created an urgent need for standardized, interoperable communication protocols capable of orchestrating complex, stateful workflows across heterogeneous agents. Traditional point-to-point (P2P) API designs and monolithic orchestration platforms lack the modularity, extensibility, and observability required to scale dynamic, decentralized agent ecosystems. The Agent-to-Agent (A2A) protocol emerges as a foundational solution by leveraging HTTP, JavaScript Object Notation - Remote Procedure Call (JSON-RPC), and Server-Sent Events (SSE) to enable discoverable capabilities, task-driven workflows, structured artifact exchanges, and real-time status updates. This review article provides a comprehensive analysis of the A2A protocol, articulating its core architectural principles—including the role of *Agent Cards*, *Tasks*, *Messages*, *Parts*, and *Artifacts*—as well as the transport, authentication, and event-driven mechanisms that underpin secure and auditable agent interoperability. We examine a broad range of industrial and research use cases, spanning enterprise automation, smart manufacturing, supply chain orchestration, personalized education, and disaster management. Furthermore, we critically evaluate deployment challenges such as schema evolution, multi-tenant authentication, durable state persistence, semantic interoperability, and privacy preservation. Drawing from these insights, we propose prescriptive design patterns and best practices to ensure robust, scalable A2A system implementations. Finally, we highlight future research directions, including decentralized discovery, confidential computing, and AI-driven orchestration, positioning A2A as a keystone for building resilient, composable agentic networks across domains.**

***Index Terms*—Agent-to-Agent Protocol, Autonomous Multi-Agent Systems, Interoperability, Task-Oriented Coordination, Event-Driven Workflow, Secure and Scalable Meshes**

I. Introduction

The proliferation of autonomous agents across distributed systems, cloud environments, and edge networks has introduced significant challenges to interoperability, scalability, and system resilience [1], [2]. Traditional architectures, which relied heavily on P2P Application Programming Interfaces (APIs), rigid integration contracts, or monolithic orchestration layers, have become increasingly brittle in the face of dynamic, heterogeneous agent ecosystems [3], [4]. Existing systems often struggle with ad-hoc discovery mechanisms, fragmented identity management, inconsistent authorization models, and brittle state management strategies [5], [6], [7]. Furthermore, as the number and diversity of agents grow, ensuring secure, audit-friendly, and privacy-compliant collaboration among agents becomes an operational and architectural imperative [8], [9]. These challenges are magnified in environments that demand real-time responsiveness, modular extensibility, decentralized control, and long-lived, stateful interactions spanning multiple organizational boundaries [10], [11].

P.P. Ray is with Department of Computer Applications, Sikkim University, Sikkim, India. Email: ppray@cus.ac.in

Against this backdrop, there is a clear motivation to develop a standardized protocol that enables agents—regardless of vendor, domain, or deployment environment—to seamlessly discover, interact, and collaborate in a secure, stateful, and extensible manner [12], [13], [14]. The motivation behind the A2A protocol is rooted in addressing these systemic deficiencies by reimagining inter-agent communication as a first-class construct, not an afterthought layered on top of existing APIs [15], [16]. A2A aims to provide a universal, transport-agnostic framework that abstracts core agentic workflows—task delegation, artifact exchange, state transitions, and notification delivery—into a composable, machine-readable, and resilient protocol architecture [17], [18], [19].

The philosophy underpinning the A2A protocol departs fundamentally from conventional Remote Procedure Call (RPC) or API gateway models [20], [21]. Rather than treating agent as passive service endpoints, A2A recognizes them as autonomous participants with discoverable capabilities, dynamic state transitions, and structured outputs [22], [23]. Central to this philosophy is the notion that every agent should advertise its competencies (i.e. skills) via standardized, machine-readable *Agent Cards*, enabling programmatic discovery and capability negotiation [24]. *Tasks* are treated as durable, auditable contracts between agents, with every exchange of information—whether it be structured data, binary *Artifacts*, or media streams—captured as typed *Parts* and *Artifacts*, preserving provenance and enabling rich introspection. By leveraging mature, widely adopted transport protocols like HTTP, JSON-RPC 2.0, and SSE, A2A ensures compatibility with existing network infrastructure while introducing powerful abstractions for streaming, push notifications, and error handling. Table I presents abbreviations used in this article.

This article is motivated by three primary objectives:

- To analyze the architectural foundations, key constructs, and operational semantics of the A2A protocol, elucidating its innovations relative to traditional multi-agent communication models.
- To critically evaluate the deployment challenges and operational considerations—such as schema evolution, state durability, secure multi-tenant authorization, semantic interoperability, and privacy preservation—that practitioners must navigate in real-world A2A deployments.
- To synthesize and propose a set of prescriptive best practices, deployment patterns, and research directions

that advance the state-of-the-art in scalable, secure, and composable agent-based systems.

The key contributions of this review article are fourfold:

- Presents a concise exposition of the A2A protocol's core abstractions—*Agent Cards, Tasks, Messages, Parts*, and *Artifacts*—and their integration with HTTP/JSON-RPC transports, authentication schemes, and notification mechanisms.
- Surveys early industry and research implementations, demonstrating A2A's impact on workflows in DevOps, manufacturing, supply chains, healthcare, education, and emergency response.
- Analyzes real-world deployment challenges, including schema evolution, secure discovery, durable state persistence, multi-modal content negotiation, and edge-device constraints.
- Outlines a forward-looking roadmap detailing decentralized discovery, confidential computing enclaves, privacy-preserving payloads, semantic ontology alignment, AI-driven orchestration, and mesh-aware optimizations.

The paper is organized to build understanding from fundamentals to advanced deployment. Section II reviews foundational research and early A2A efforts. Section III details A2A's core constructs, transports, and security model. Section IV presents state-of-art in terms of industry adoption surveys use cases and real-world implementations. Section V Deployment Challenges analyzes schema evolution, discovery, security, and reliability hurdles. Section VI presents future Directions. Section VII concludes the article.

II. Related Works

The concept of agent interoperability has evolved substantially over the past decades, building upon foundational work that spans multiple facets of multi-agent systems (MAS). Langley et al. [25] initially addressed the essential capability of infrastructure discovery in MAS, creating adaptive mechanisms that allowed agents to dynamically recognize and respond to evolving network topologies. This concept of dynamic adaptation was crucial in allowing agents to seamlessly interoperate within P2P infrastructures. Further advancements have emerged through the efforts of Sohn et al. [26], [27], who developed the A2X dataset specifically designed to tackle the nuanced challenges of agent-to-environment interactions. By incorporating sophisticated metrics—termed "multiverse" metrics—they overcame previous evaluation limitations, offering deeper insights into trajectory prediction scenarios, particularly emphasizing interactions that traditional benchmarks inadequately represented. The introduction of self-supervised learning techniques by Bhattacharyya et al. [28] offered additional robustness to agentic systems, particularly through Secure Sockets Layer (SSL)-Interactions, an approach using specialized pretext *Tasks* such as range gap and closest distance prediction. These *Tasks* provided a stronger semantic understanding of inter-agent dynamics, significantly improving the accuracy and interpretability of interaction predictions. Simultaneously, multi-agent reinforcement learning (MARL)

TABLE I: Alphabetical Glossary of Abbreviations

Abbreviation	**Full Form**
A2A	Agent-to-Agent Protocol
ABAC	Attribute-Based Access Control
ACME	Automatic Certificate Management Environment
API	Application Programming Interface
CBOR	Concise Binary Object Representation
CDN	Content Delivery Network
CI/CD	Continuous Integration / Continuous Deployment
CMDB	Configuration Management Database
CRL	Certificate Revocation List
CRUD	Create, Read, Update and Delete
DANE	DNS-based Authentication of Named Entities
DDoS	Distributed Denial-of-Service
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
DHT	Distributed Hash Table
ECDHE	Enforce Forward Secrecy Suites
ELK	Elasticsearch – Logstash – Kibana stack
FIPS	Federal Information Protection Standard
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IAM	Identity and Access Management
IP	Internet Protocol
ITSM	IT Service Management
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
JSON-RPC	JSON Remote Procedure Call
JTI	JWT ID
JWKS	JSON Web Key Set
JWT	JSON Web Token
LLM	Large Language Model
MARL	Multi-Agent Reinforcement Learning
MAS	Multi-Agent Systems
MCP	Model Context Protocol
MIME	Multipurpose Internet Mail Extensions
mTLS	Mutual Transport Layer Security
OCSP	Online Certificate Status Protocol
OAuth2	Open Authorization 2.0
OIDC	OpenID Connect
P2P	Peer-to-Peer
PII	Personally Identifiable Information
PKI	Public Key Infrastructure
QUIC	Quick UDP Internet Connections
RAG	Retrieval-Augmented Generation
RBAC	Role-Based Access Control
REST	Representational State Transfer
RPC	Remote Procedure Call
SaaS	Software-as-a-Service
SHACL	Shapes Constraint Language
SEV	Secure Encrypted Virtualization
SGX	Software Guard Extensions
SLA	Service Level Agreement
SLO	Service Level Object
SRE	Site Reliability Engineering
SPIFFE	Secure Production Identity Framework for Everyone
SSE	Server-Sent Events
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TTL	Time to Live
UDP	User Datagram Protocol
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
W3C	World Wide Web Consortium
WAF	Web Application Firewall

methods have seen a transformative rise in popularity, especially in cooperative positioning applications. Tedeschini et al. [29], [30] successfully framed cooperative vehicle positioning as decentralized partially observable Markov decision processes (Dec-POMDP), implementing innovative MARL algorithms like ICP-MAPPO. Their results demonstrated significant enhancements in accuracy and communication efficiency, thus laying a critical foundation for decentralized autonomous decision-making.

Parallel developments in micro-scale computing were explored by Bosse and Lehmhus [31], who innovatively employed mobile multi-agent systems integrated within robotic materials to achieve cluster-scale computations in highly resource-constrained environments. This approach emphasized the importance of adaptive algorithms for robust distributed computing, especially as devices scale down to micro and sub-micro levels. Minelli's research [32] further underscored the value of decentralized communication and coordination, proposing the CoMix framework, which allowed agents explicit decision-making capability regarding when to coordinate or act independently. Such flexibility enabled scalable and adaptive responses within complex, distributed environments.

With the announcement of Google's A2A Protocol [33], the field witnessed a significant leap forward. The A2A Protocol, distinct from its predecessors, standardized agent interactions through JSON-based *Agent Cards*, enabling streamlined capability discovery and secure multi-modal exchanges. The design principles emphasized security, extensibility, and long-duration task management, utilizing widely accepted standards such as HTTP, SSE, and JSON-RPC. Ghosh [34] extended the conceptual potential of A2A and Model Context Protocol (MCP) by applying them within engineering domains, proposing frameworks that integrate diverse legacy software tools through standardized agent communications. This holistic approach addressed the fragmented software landscape typically found in multidisciplinary engineering projects.

Practical adoption insights were also documented by A. Ghosh [35] and Mattson [36], who demonstrated seamless integration of the A2A protocol within popular development frameworks and real-world scenarios. These tutorials and implementation guides significantly reduced barriers to entry, facilitating broader industry adoption. Jellema [37] critically assessed the broader applicability of A2A beyond purely AI-driven systems, highlighting its potential as a universal interaction framework applicable to general software integrations, not limited by AI-specific features. This broader perspective increased the perceived versatility and adoption potential of A2A. Further clarity on protocol interactions was provided by Chawla [38], who visually emphasized how A2A and MCP complement rather than compete. A2A enabled inter-agent collaboration, while MCP focused on agent-tool interactions, together providing a comprehensive ecosystem for agentic workflows. BytePlus's comprehensive guide [39] articulated the real-world implications of the A2A protocol, highlighting how its standardized interactions resolved previous industry-wide issues of isolated agent operations, thus significantly improving multi-agent workflow automation and security. Mukhopadhyay [40] elaborated on Python-based open-source implementations of A2A, enhancing developer accessibility. These contributions simplified practical adoption, allowing rapid deployment and prototyping of agent-driven solutions across different technology stacks. The supportive role of an expanding partner ecosystem was emphasized by Ichhpurani [41], noting the collaborative efforts of numerous technology partners that have already begun to embed A2A within enterprise-level solutions, thereby validating its practicality and industry relevance. Comprehensive industry analysis by VentureBeat [42] and The New Stack [43] echoed the transformative potential of Google's A2A protocol, particularly noting its alignment and complementary nature with MCP, its community-driven evolution, and its ability to securely manage complex, asynchronous *Tasks*. These analyses highlighted A2A's potential as a universal standard for future multi-agent interoperability, marking a significant step forward from previously fragmented agentic communication practices. Collectively, this body of work establishes a robust foundation upon which the current review builds, offering a novel, integrative perspective on the evolutionary trajectory of multi-agent interoperability, specifically through the lens of the recently launched A2A protocol. This review uniquely synthesizes historical insights with emerging capabilities, clearly delineating A2A's innovative contributions and outlining the trajectory for future developments in the agentic ecosystem. Table II presents comparison of related works.

Lessons Learned

Reviewing the existing literature reveals several insights and gaps that underscore the necessity of standardization in agent-to-agent communication. Historically, agent interactions within MAS have been explored primarily through domain-specific and isolated frameworks, as demonstrated by early research, where discovery mechanisms were confined to P2P local networks without global standards. Moreover, works such as the A2X dataset studies highlighted critical shortcomings in the current evaluation protocols and emphasized the necessity for robust interaction modeling to handle multimodal agent-environment scenarios, yet they did not address standardization for cross-platform interoperability. Similarly, SSL-Interactions and cooperative positioning research illuminated the benefits of enhanced agent interaction, specifically advocating for standardized protocols to effectively model complex real-time interactions. Despite these contributions, each study was constrained to specific contexts or datasets, lacking a universally applicable protocol. Further, existing contributions have largely addressed agent-to-agent communication indirectly or through highly specialized application scenarios such as robotic materials integration, decentralized coordination, and engineering design interoperability. These efforts demonstrated the clear value and necessity of unified, standardized approaches but still maintained isolated implementations, not offering generalizable frameworks beyond their niche areas. The recent launch of Google's A2A protocol has emphasized industry-wide recognition of this standardization gap. By introducing open standards for secure, structured, and modality-agnostic agent interactions, A2A demonstrates the feasibility and critical need for cross-vendor interoperability in real-world enterprise and industrial applications. These cumulative

TABLE II: Comparative Summary of Related Works

Article	Contribution	Technology/Protocol	Limitation	Key Aspects
[25]	Discovery mechanism for agent infrastructures	P2P (Gnutella) MAS Discovery	Limited to local/nearby network updates	Early MAS discovery, dynamic infrastructure learning
[26]	A2X dataset for multimodal human trajectory prediction	Human Trajectory Prediction Dataset	Limited real-world environment generalization	Multiverse metrics, environment-agent interaction
[27]	Agent-environment interaction benchmark (A2X)	A2X benchmark (motion prediction)	Simulated scenarios, not full real-world complexity	Improved metrics for multimodal HTP models
[28]	SSL-Interactions: Self-supervised *Tasks* for interactions	Pretext *Tasks* in Trajectory Prediction	Focused mainly on vehicle interactions	Range-gap prediction, direction prediction *Tasks*
[29]	Cooperative positioning via MARL	Multi-Agent Proximal Policy Optimization (MAPPO)	Assumes CARLA simulated environment	Decentralized, dynamic inter-vehicle communication
[30]	Distributed MARL for vehicular positioning	ICP-MAPPO (Bayesian filtering + MARL)	Non-Gaussian scenarios only partially addressed	Scalability for communication-efficient positioning
[31]	Self-adaptive robotic materials via MAS	Mobile Agents (JavaScript JAM)	Very low-resource hardware challenges	Cluster computing inside robotic materials
[32]	Coordination strategies in decentralized MARL	CoMix Strategy (Coordination choice)	Coordination overhead in scaling	Explicit agent independence vs cooperation
[33]	Introduction to Google's A2A Protocol	A2A Protocol (open, agent interoperability)	Early stage, needs community-driven expansion	Multi-agent collaboration across vendors
[34]	Engineering Design MAS via A2A + MCP	Legacy Tool Wrapping, MCP, A2A	Conceptual framework, not yet deployed	Engineering workflow optimization via agentic design
[35]	Overview of Google's A2A architecture	A2A + MCP hybrid usage	Lack of large-scale industrial case studies	Simplified multi-agent interoperability guide
[36]	Integrating Semantic Kernel Python with A2A	Semantic Kernel Agents + A2A Interoperability	Early integration, experimental phase	Plug-and-play Azure OpenAI agents via A2A
[37]	Clarification: A2A is broader than just AI agents	A2A protocol generalized for software agents	Perceived AI-centric marketing confusion	Highlights protocol's utility for traditional software too
[38]	Visual guide to A2A (non-competing with MCP)	A2A-MCP cooperation framework	Surface-level guide (missing technical depth)	Clear division: A2A (agent-to-agent) vs MCP (agent-to-tool)
[39]	Comprehensive tutorial on A2A interoperability	A2A architecture + real use cases	Still early adoption phase in enterprises	Explains shipping container analogy for A2A standardization
[40]	Python A2A implementation overview	Open-source A2A (Python library)	Implementation maturity evolving	Democratization of multi-agent A2A setup
[41]	Google's Partner Ecosystem around Agentic AI + A2A	Partner adoption of A2A at Google Cloud Next	Mostly industry news, little technical detail	Real-world traction with 1000+ agent use-cases
[42]	VentureBeat coverage of Google's A2A protocol	Industrial positioning of A2A vs AGNTCY, MCP	Ecosystem competition risk (fragmentation)	Open standardization push, enterprise-grade security
[43]	New Stack article on A2A announcement	A2A complementing MCP, technical overview	Community contribution roadmap unclear yet	Highlights real-time feedback, SSE usage in A2A

lessons underscore a clear motivation: a comprehensive and unified review of the emerging A2A protocol is necessary to contextualize its broader implications, identify existing knowledge gaps, and provide foundational guidance for future research and practical deployments.

Novelty of This Review

This paper represents the first comprehensive and systematic review of Google's newly introduced A2A protocol, distinctly addressing a gap not covered by any prior research. Unlike previous studies, which either focus narrowly on domain-specific applications or broadly discuss multi-agent systems without standardized frameworks, this paper uniquely synthesizes existing knowledge around the A2A protocol from a multidisciplinary viewpoint. It critically analyzes its architectural principles, security mechanisms, modality flexibility, and task lifecycle management. Furthermore, by meticulously comparing the A2A protocol with existing agent communication methodologies, such as the MCP and other legacy multi-agent frameworks, the paper delivers a deep technical understanding of how A2A extends, complements, and innovates beyond traditional approaches. Through rigorous literature synthesis, the review identifies A2A's transformative potential in agent interoperability and pinpoints specific areas—like agent discovery, dynamic capability negotiation, and multimodal agent coordination—that have received minimal attention in previous research. Moreover, this review provides novelty through its holistic approach, explicitly examining A2A protocol's implications across varied real-world scenarios including enterprise integration, healthcare automation, intelligent transportation systems, and robotic applications. While previous articles have only superficially touched upon A2A or discussed its general applicability, this review systematically evaluates the protocol's strengths and challenges from theoretical, practical, and strategic perspectives. The analysis further explores emerging themes such as governance, adoption challenges, and community-driven standard evolution, areas scarcely addressed in current literature. Thus, the review serves as both a foundational reference and a roadmap for researchers, developers,

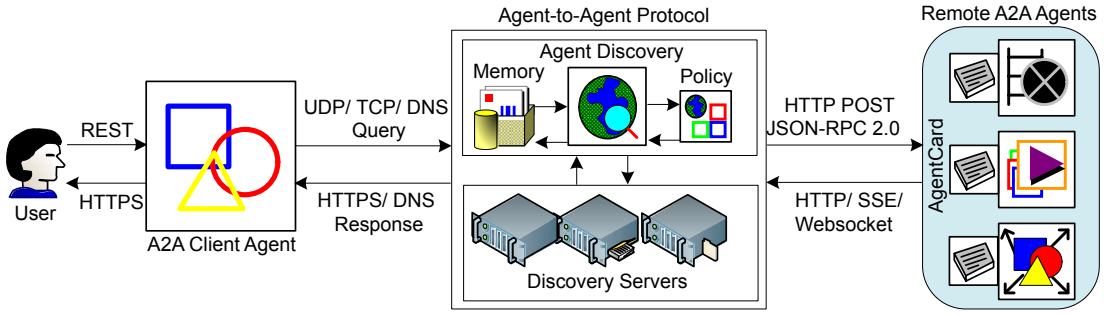

Fig. 1: Overview of A2A architecture

and enterprises aiming to leverage or contribute to the A2A ecosystem, significantly enriching the academic discourse and guiding future development toward a unified and interoperable multi-agent future.

III. Overview of the A2A Protocol

A. A2A Protocol Foundation

At its core, the A2A protocol establishes a robust, web-native foundation for enabling heterogeneous autonomous software agents to collaborate seamlessly [20]. The Overview of A2A positions it not merely as another API layer but as a full-fledged interoperability standard that sits atop proven internet technologies—namely HTTP, JSON-RPC 2.0 and SSE. By adopting these ubiquitous transports and encodings, A2A benefits from existing enterprise toolchains (e.g. load balancers, proxies, observability platforms) and security appliances without reinventing low-level networking or serialization formats [44]. Instead, it focuses on the higher-order problem of orchestrating distributed, stateful *Tasks* among opaque agents that encapsulate their own reasoning, tool integrations and user-interaction logic.

1) Actors: The Actors in A2A define the three distinct roles in every transaction. A2A prevents responsibility creep such as (i) clients handle context and user flow, (ii) agents focus exclusively on domain expertise, and (iii) users remain insulated from underlying inter-agent mechanics. Taken together, these protocol-level components ensure that A2A deployments integrate effortlessly with existing cloud-native and on-premises stacks. Developers and architects gain a consistent method for connecting agents, orchestrating multi-step workflows and maintaining end-to-end security—all while leveraging battle-tested web standards instead of learning yet another bespoke transport or serialization format.

- User
  The User may be a human application or another service invoking an agent to fulfill a concrete need.
- Client
  The Client is the on-behalf-of component—often an agent itself or a front-end service—that translates user intent into A2A calls, manages session context and renders results.
- Server (Remote Agent)
  Finally, the Remote Agent embodies the "*black-box*" executor - it exposes its capabilities via well-known endpoints, ingests *Tasks*, executes business logic or Large Language Model (LLM) reasoning pipelines, and returns structured *Artifacts* and status *Messages*. Table III presents detailed responsibilities for each of he actors in A2A protocol.

In the A2A ecosystem, a user's high-level request (e.g. "Generate a quarterly sales chart") is first received by an A2A Client Agent, which orchestrates discovery and execution. To find capable collaborators, it uses Domain Name System (DNS) or an enterprise registry to locate each remote agent's well-known *Agent Card* (a small JavaScript Object Notation (JSON) manifest describing skills, endpoint Uniform Resource Locators (URLs), I/O formats, and authentication requirements). Once the client has fetched one or more *Agent Cards*—via direct DNS-based queries, a curated corporate catalog service, or a protected private-API gateway—it knows exactly which agents can handle which sub*Tasks*. The client then dispatches a stateful Task object over JSON-RPC 2.0 to the chosen remote A2A Agent, optionally opening an SSE/WebSocket stream or supplying a webhook URL for real-time updates. Each Task carries unique IDs, lifecycle status (submitted, working, input-required, completed, etc.), and accumulates *Messages* and *Artifacts* (text, files, structured data) in "*Parts*." By cleanly decoupling discovery (*Agent Cards*) from communication (HTTP/JSON-RPC, SSE, webhooks) and object semantics (*Task/Message/Artifact*), A2A enables dynamic, multi-agent workflows that can discover, negotiate, and execute complex operations across disparate systems—using nothing more exotic than standard web plumbing. Figure 1 presents the basic overview of A2A architecture.

2) Transport: Underpinning all interactions is the Transport mechanism. A2A adopts HTTP as its canonical carrier, leveraging Representational State Transfer (REST)-style endpoints for JSON-RPC method calls (e.g. *Tasks/send*, *Tasks/get*, *Tasks/cancel*, etc.). This choice guarantees compatibility with firewalls, enterprise proxies and monitoring infrastructure. When *Tasks* span significant duration or require streaming partial outputs—such as real-time data analysis or multimedia generation—A2A clients may invoke *Tasks/sendSubscribe* to establish an SSE connection. The server then emits *TaskstatusUpdateEvent* and *TaskArtifactUpdateEvent* records as deltas, enabling clients to update UI components or downstream pipelines incrementally. In disconnected or mobile scenarios, callback webhooks registered via *Tasks/pushNotification/set permit* fully asynchronous delivery of final Task

TABLE III: Actors and Their Responsibilities in A2A Protocol

Actor	Responsibility
User	A human or service that initiates a request ("Generate an image of a sunset"). Interacts via UI, CLI, or another front-end.
A2A Client	The local "host agent" code that: • Translates user intent into A2A RPC calls (*Tasks/send*, *Tasks/sendSubscribe*, etc.) • Packages content into *TasksendParams* • Listens for streaming SSE events or polls via *Tasks/get* • Receives webhook POSTS if configured • Renders statuses, prompts for more input, and final *Artifacts* back to the User.
A2A Server	The remote, opaque agent implementation that: • Exposes a JSON-RPC HTTP endpoint • Reads incoming *TasksendParams* → runs its internal "TaskLogic" (LLM calls, tool invocations, multi-turn dialogue) • Emits *TaskstatusUpdateEvent* and *TaskArtifactUpdateEvent* over SSE or returns a final *Task* result • Optionally POSTs back to the client's webhook URL as a *Task* snapshot.

payloads, with robust retry semantics managed by the agent's notification service.

3) Authentication and Authorization: No enterprise protocol can ignore security, and authentication and authorization in A2A reflect production-grade safeguards. Agents declare supported auth schemes— Open Authorization 2.0 (OAuth2), API keys, or Mutual Transport Layer Security (mTLS)—in their *Agent Card* metadata, but never exchange user or client credentials inline. Instead, clients obtain tokens through standard identity platforms (e.g. OpenID Connect (OIDC), Secure Production Identity Framework for Everyone (SPIFFE)), then attach them as HTTP headers on each JSON-RPC request. Agents validate these headers against their configured trust anchors, rejecting unauthorized calls with HTTP 401/403 responses and challenge headers (e.g. WWW-Authenticate). Where *Tasks* require elevated privileges or use of sensitive tools, a remote agent can transition a *Task* into an input-required state, embedding an authentication structure that instructs the client which additional credentials to procure out-of-band. By keeping credential management external to the A2A payload, the protocol remains clean, audit-friendly and highly resistant to replay or injection attacks.

B. Agent Card

1) Foundation of Agent Card: Within the A2A framework, each autonomous agent introduces itself to the ecosystem using an *Agent Card* — a structured JSON document that serves as a self-contained description of the agent's public interface. Fundamentally, the *Agent Card* conveys essential metadata, including a readable *name*, a succinct *description* summarizing the agent's capabilities, and a *url* indicating the endpoint where JSON-RPC interactions are accepted. Together, these fields allow client agents to discover not just *where* to direct communications but also *what* functionality an agent provides.

Beyond these primary identifiers, an optional *provider* field offers additional context by specifying the organization responsible for the agent, accompanied by a reference URL for branding or informational purposes. Version management is handled through a flexible *version* attribute, enabling agents to signal their software release state, which is useful for maintaining compatibility across upgrades. When additional technical material, such as integration guides or API references, is available, the agent may include a *documentationUrl* to support deeper client-side integrations.

An *Agent Card* also declares operational capabilities through a dedicated *capabilities* section, where boolean values specify whether the agent supports features like real-time streaming updates, asynchronous push notifications, or tracking of state transitions. Advertising these capabilities allows clients to adapt their communication strategy accordingly, preferring streaming for low-latency use cases or relying on push-driven updates when event-based operation is preferable.

Security configurations are explicitly outlined under the *authentication* field, listing acceptable authentication schemes such as Bearer tokens or Basic HTTP authentication. Instead of embedding sensitive data directly, the *Agent Card* specifies the mechanisms clients must use to authenticate securely, preserving clean separation between protocol metadata and credential exchange, and allowing seamless integration with existing identity management infrastructures like OAuth 2.0, SSO, or mTLS.

To facilitate interoperability, the *Agent Card* defines acceptable data formats through *defaultInputModes* and *defaultOutputModes*, each listing supported MIME types. These defaults ensure clients can quickly verify compatibility—whether exchanging simple text prompts, structured JSON payloads, or more complex media formats such as images or video—without needing to inspect individual skills initially.

At the core of the *Agent Card* lies the *skills* array, wherein each skill represents a specific capability offered by the agent. Each skill object includes a unique *id* for programmatic reference, a readable *name*, and a concise *description* to guide client interpretation. Skills are further classified using *tags*, enabling semantic filtering across large agent ecosystems. Optional *examples* offer practical prompts to illustrate usage patterns, assisting developers and end-users alike. Skills may also override global defaults for input and output formats if necessary, tailoring interaction types based on task-specific requirements. Table IV summarizes the key attributes of the *Agent Card*.

2) Agent Discovery: In a distributed ecosystem of autonomous agents, the ability to locate and engage collaborators dynamically is fundamental. Agent discovery in the A2A framework revolves around the publication and consumption of *Agent Cards*—self-descriptive JSON documents that reveal

TABLE IV: Attributes of the Agent Card Structure

Attribute	Detailed Description
name	The human-readable name of the agent, intended for display to users or other agents. It provides a recognizable identifier, e.g. "Recipe Agent".
description	A textual explanation that summarizes the agent's functionality and role. Helps users or calling systems understand the agent's capabilities at a glance.
url	The endpoint address (typically HTTPS) where the agent is hosted and can be contacted for performing operations.
provider	(Optional) Metadata about the agent's service provider, including the organization's name and official URL. Useful for trust verification and branding.
version	Specifies the agent's version, following any provider-defined scheme (e.g. "1.0.0", "2023.09"). Important for compatibility and feature negotiation.
documentationUrl	(Optional) A link to external documentation offering detailed technical instructions, API usage guidelines, or tutorials.
capabilities	A set of optional Boolean flags indicating advanced behaviors the agent supports, such as SSE (streaming), push notification capabilities, and state transition tracking for task workflows.
authentication	Defines required authentication mechanisms, including supported schemes (e.g. Basic, Bearer tokens) and optional credentials for accessing private *Agent Cards*. It aligns with OpenAPI[1] authentication structures.
defaultInputModes	A list of MIME types that the agent accepts as default input formats across all its skills (e.g. *application/json*, *text/plain*).
defaultOutputModes	A list of MIME types that the agent outputs as defaults (e.g. *text/plain*, *application/json*), unless overridden at the skill level.
skills	A list of modular units of capability (skills) the agent can perform. Each skill has its own name, description, input/output types, tags, and examples. Enables fine-grained discovery and task assignment.

an agent's endpoint, capabilities, authentication requirements, and supported content modalities. Rather than relying on hard-coded registry entries or manual configuration, A2A encourages multiple discovery modalities that scale from open internet-style lookups to tightly controlled private registries. Each approach strikes a different balance between accessibility, governance and security, enabling organizations to tailor agent onboarding to their operational needs.

- Open Discovery
  The simplest and most transparent mechanism is open discovery, which leverages a well-known URL convention. Agents make their *Agent Card* accessible at a standardized path—*https://<agent-domain>/.well-known/agent.json*—so that any client capable of resolving the domain via DNS can fetch the card with a straightforward HTTP GET request. This pattern reduces discovery to two steps: (i) identify a domain hosting the agent's service and (ii) retrieve the manifest at the well-known location. Because it aligns with the existing ".well-known" approach adopted by numerous internet protocols (e.g. webfinger[2], security.txt[3]), open discovery benefits from broad compatibility with web crawlers, search engines and off-the-shelf HTTP clients. Developers can bootstrap new ecosystems by configuring DNS records alone, while dynamic services—such as container-orchestrated microservices—can automatically advertise new agents by publishing valid *Agent Cards* at the well-known path.
- Registry-Based Discovery
  For enterprises that require more curated control over which agents are deemed trustworthy or suitable, a registry-based discovery model offers a middle ground. Under this pattern, agents are registered in an internal *catalog* managed by an administrator or a governance team, rather than being publicly exposed on the internet. The registry might expose a RESTful API or a dashboard interface where agents can be listed, tagged, versioned and assigned metadata such as compliance labels or usage quotas. Client applications then query the registry—via API calls authenticated through enterprise IAM—for a list of approved *Agent Cards*. This curated process enables organizations to enforce policies around certification, deprecation timelines and domain-specific compliance, while still preserving the automated lookup semantics of A2A. New agents are onboarded through a controlled process, which might include schema validation, security audits and manual review, before their cards are propagated to the enterprise catalog. The registry thus serves as a single source of truth for sanctioned agents, reducing the risk of rogue or outdated components entering production workflows.
- API-Based Discovery
  In scenarios where agent ecosystems must remain entirely private—perhaps due to proprietary algorithms or sensitive data contexts—a private discovery API model is preferred. Rather than exposing *Agent Cards* over the public internet or in a semi-trusted enterprise catalog, the agent platform may offer custom API endpoints that deliver *Agent Cards* only to authenticated clients. These private "agent stores" could be backed by corporate service meshes, API gateways or on-premises key-value stores. Clients present bearer tokens, mutual TLS certificates or other credentials to the discovery endpoint, which then serves tailored *Agent Card* content based on the client's identity and authorization level. This mechanism enables fine-grained segmentation—different teams or business units see subsets of agents relevant to their domain, and specialized agents can be hidden from all but the most privileged clients. Because private discovery often relies on bespoke APIs, A2A leaves the implementation details to the organization, focusing instead on the semantics of *Agent Card* consumption rather than mandating a specific registry schema. Table V briefs various discovery mechanisms.

3) Securing Agent Card: Regardless of how an *Agent Card* is located, securing the discovery process is paramount. *Agent Cards* may contain sensitive details—endpoints that should remain hidden from competitors, tags that reveal internal

[2]https://webfinger.net/
[3]https://securitytxt.org/

TABLE V: Agent-Card Discovery Options in the A2A Protocol

Discovery Mode	How it Works	Where the Service Lives	What the Call Yields
Open / Direct	A2A Client already knows (via config, DNS SRV, service mesh, etc.) a domain for the target agent. It issues *GET https://<domain>/.well-known/agent.json* and parses the JSON. No central catalogue is involved.	Same host or public domain that runs the remote A2A server (external Software-as-a-Service (SaaS), partner, or internal micro-service).	*One* complete *AgentCard* document for that single agent.
Registry-Based	A2A Client queries an enterprise "Agent Catalog" REST endpoint, e.g. *GET https://registry.company.com/agents/catalog*. The registry responds with an array of Agent-Card URLs that have passed governance checks. A2A Client then HTTP-GETs each URL to obtain the individual cards.	Enterprise-controlled registry service—typically in a corporate VPC, private cloud, or approved SaaS catalogue (e.g. Backstage, ServiceNow).	A *list of URLs*; each URL subsequently returns a normal *AgentCard*.
Private-API	For confidential or tenant-specific agents the A2A client calls a protected API such as *GET https://internal.api/agents/{id}* (mTLS/OAuth required). The API looks up the requested agent internally and returns its card payload directly.	Strictly inside the organisation's private network (on-prem DC or private cloud VPC) behind firewall / IAM controls.	One *AgentCard*; usually contains sensitive skills or auth metadata that must not appear in public or registry listings.

workflows or even placeholder credentials for special integration flows. To prevent unauthorized clients from harvesting this information, implementers may enforce transport-layer security (i.e. HTTPS with strong TLS ciphers), require mutual TLS authentication for the discovery endpoint or integrate discovery into existing IAM systems. In open discovery scenarios, organizations can still protect cards by deploying them behind reverse proxies that enforce IP-whitelisting or OAuth scopes, ensuring that only sanctioned clients can retrieve them. In registry and private API models, the discovery endpoint inherently enforces access control, returning different *Agent Card* variants based on the authenticated caller. Furthermore, because *Agent Cards* can embed credential hints—such as API key formats or OAuth scopes—it is essential that such sensitive entries never be exposed without authentication. A2A recommends that secret or token references within the card be encrypted or represented as opaque identifiers that require a secondary exchange to resolve. This two-step pattern—fetch card metadata, then perform a separate authenticated call to retrieve actual credentials—ensures that mere possession of the *Agent Card* does not grant unintended privileges. Additionally, audit logs should record every discovery request, capturing the client identity, timestamp and card version returned, thereby providing comprehensive traceability for compliance and forensic investigations. By offering a flexible suite of discovery methods—ranging from open, DNS-based lookups to highly controlled private APIs—A2A empowers organizations to architect agent ecosystems that meet their unique security, compliance and governance requirements. The standardized *Agent Card* format underpins this versatility, ensuring that once a client has located a card, it can consistently parse endpoint URLs, supported authentication schemes and content modalities. As the A2A community evolves, further innovations—such as decentralized registries, secure multi-party discovery protocols or federated catalog services—can layer on top of this foundation, enriching agent interoperability while preserving the core promise: seamless, automated discovery of intelligent services in an ever-expanding landscape of autonomous collaborators.

C. Core Objects of A2A Protocols

The A2A protocol centers around a handful of well-defined, reusable data structures—collectively known as its Core Objects—that enable client agents and remote agents to coordinate complex, multi-step workflows in a reliable, extensible manner. These objects encapsulate both the state and the content of agent interactions, while keeping the protocol agnostic to any particular domain or implementation technology. In this discussion, we delve deeply into each Core Object—*Task, Artifact, Message, Part*, and the *Push Notification* infrastructure—unpacking their schemas, life cycles and the semantics they carry.

1) Task: In A2A, a *Task* encapsulates the full context of an interaction between client and agent, acting as the container for *Messages*, state, and results. When a client invokes *Tasks/send* or *Tasks/sendSubscribe*, it supplies a client-generated Universally Unique Identifier (UUID) as the *Task*'s id, guaranteeing idempotent processing: retries with the same id will not spawn duplicate executions. If multiple related requests belong to the same conversational thread—say, a multi-step provisioning workflow—a shared sessionId groups them, enabling the agent to maintain continuity across discrete *Task* objects. Internally, agents use the *Task*'s metadata field to store routing hints or Service-level agreement (SLAs), such as "priority: high" or "region: us-east-1," which orchestration layers can leverage for load balancing or cost accounting. *Tasks* persist on the agent side according to retention policies, meaning clients can even reconnect days later and fetch or resume the conversation as long as the *Task* remains within the agent's storage window.

A *Task*'s status provides a snapshot of its current phase and any instruction the client must heed to progress. The state property moves through a well-defined finite state machine: submitted upon receipt, working while processing, input-required if human or machine input is needed, and one of the terminal states—completed, failed, or canceled. Each transition can carry a nested message, itself a Message object, to supply human-readable or structured guidance. The timestamp adheres to ISO 8601[4] formatting, which is crucial for correlating logs across distributed systems and enforcing SLA timers.

[4]https://www.din.de/de/mitwirken/normenausschuesse/nia/entwuerfe/wdc-beuth:din21:266843241

Agents may also include diagnostic codes or performance metrics in an extended metadata map inside *Taskstatus*, such as estimated remaining time or CPU utilization, empowering clients to make load-management decisions.

For real-time monitoring of long-running *Tasks*, A2A leverages SSE. Upon calling *Tasks/sendSubscribe*, the client opens an SSE stream over HTTP; the agent then emits *TaskstatusUpdateEvent* objects whenever the *Task*'s state changes. Each event payload re-affirms the *Task* id, provides the updated *Taskstatus*, and flags final: true only when the *Task* reaches a terminal state, signaling the client to close the stream. To ensure reliable delivery, events include optional per-event metadata—for example, a sequence number to detect dropped *Messages* or a retry hint for back-pressure management. Well-designed clients use the SSE *Last-Event-ID* header to resume interrupted streams without losing updates.

Parallel to status events, agents send *TaskArtifactUpdateEvent Messages* when new result fragments become available. Each update contains the *Task* id, a single *Artifact* (or an appended *Part* thereof), and optional sequencing metadata. Streaming *Artifacts* in small, ordered chunks allows clients to begin processing or displaying partial outputs—such as progressively rendering a large image or incrementally building a JSON report—rather than waiting for the entire payload. The agent marks the final *Part* of an *Artifact* with lastChunk: true, ensuring that clients know when they've received the complete deliverable. If a client disconnects and later resubscribes, the server can replay missed I or allow the client to fetch the entire *Artifact* via *Tasks/get*.

Clients drive *Task* creation and continuation through the *TasksendParams* structure. Beyond id and message, it accepts an optional *sessionId* to tie back to ongoing threads, *historyLength* to request recent chat context, and a *pushNotification* sub-object for offline callbacks. Embedding *pushNotification* inline lets clients atomically set up webhook-based notifications without a separate RPC. Custom metadata fields—such as Correlation IDs or feature flags—can inform remote agents or enterprise middleware how to route or transform incoming requests. Well-architected clients ensure that message.*Parts* carry sufficient context: a text prompt, any file attachments, and schema validation hints, enabling agents to validate inputs before deep processing begins.

The *Taskstate* enumeration codifies every supported phase of a *Task*'s life cycle. Implementers typically model the state machine as:

- submitted $\rightarrow$ upon initial request
- working $\rightarrow$ when processing begins
- input-required $\rightarrow$ if disambiguation or credentials are needed
- completed, failed, or canceled $\rightarrow$ terminal states Clients and agents must honor valid transitions (e.g. user cannot move from completed back to working). If an agent cannot parse the client's request or encounters an unexpected error, it should transition to failed with an explanatory message. In fire-and-forget scenarios, a client might cancel orphaned *Tasks* via *Tasks/cancel*, prompting a canceled transition to release resources.

2) Artifact: Artifacts are the durable outputs of *Tasks*. Each *Artifact* can carry an optional name and description—for human or pipeline discovery—and a map of metadata for retention policies, content hashes, or semantic tags (e.g. "report-Q2"). The *Parts* array holds ordered *Part* objects; when agents support streaming, they set append: true on successive updates and lastChunk: true on the final fragment. The index field disambiguates multiple *Artifacts* produced by the same *Task*—common in workflows that generate separate PDF and CSV exports. Clients store *Artifacts* in local caches or cloud storage, indexing them by i id and $artifact$ index for efficient retrieval.

3) Message: Messages carry all conversational and control data outside of final *Artifacts*. A Message's role—"user" or "agent"—distinguishes sender intent, while its *Parts* array holds the actual payload fragments. Agents send *Messages* to report partial progress, ask follow-up questions, or negotiate UI modalities, whereas clients send *Messages* to submit new inputs or respond to input-required prompts. Message-level metadata can include headers for tracing (e.g. traceparent for World Wide Web Consortium (W3C) trace context), priority hints or routing tags in multi-tenant environments. By archiving *Messages* in the *Task* history, both sides maintain an auditable record of the entire exchange.

4) Part: The *Part* abstraction decomposes complex payloads into typed units as mentioned later. Each Part's metadata map can advertise expected schemas, UI hints (e.g. widget: dropdown), or security labels. During inputs, clients might flag a *Part* with a JSON Schema; the agent then validates the incoming *Part* against it, ensuring type safety before processing.

- TextPart
  Raw UTF-8 content in the text field, used for prompts, markdown or logs.
- FilePart
  Encapsulates binary or structured file data. The file object may embed bytes as base64 or supply a URI for off-line retrieval, accompanied by a mimeType and optional name. This supports large attachments without bloating JSON RPC frames.
- DataPart
  Carries arbitrary JSON objects in data, enabling form submissions or schema-validated structures.

5) Push Notifications: In many enterprise scenarios, *Tasks* initiated by AI agents can span minutes, hours, or even days. Consider a workflow to ship a prototype sample across continents: (i) it involves multiple handoffs, (ii) external systems, and (iii) unpredictable delays. Relying solely on synchronous requests or polling would either tie up resources or introduce latency and complexity. To address this, A2A provides a robust push-notification subsystem that complements persistent streaming channels and ensures clients remain informed of *Task* progress—even when disconnected. Table VI demonstrates key stages of push notifications activities.

- Connected Updates
  When a client maintains an open channel to an agent—typically via HTTP plus SSE—the two sides can

TABLE VI: Push-Notification Stages for A2A

Stage / Mechanism	Initiated By	Direction	Transport	When / Why	Essential Steps & Security Hooks
Connected Streaming	Either side (usually Client via *Tasks/sendSubscribe*)	Bi-dir (status/events)	HTTP + SSE	While UI is online; get millisecond-level updates.	• Server pushes *TaskstatusUpdateEvent* / *TaskArtifactUpdateEvent*. • If Transmission Control Protocol (TCP) breaks, Client can resume with *Tasks/resubscribe*.
Disconnected Push (webhook)	Client (declares PushNotificationConfig)	Uni (Server → Client)	HTTPS POST	Mobile/back-office apps that may sleep or hide.	• Client sets config in *Tasks/send* or *Tasks/pushNotification/set*. • Server POSTs full Task whenever state ∈ {completed, input-required, failed, ...}.
PushNotificationConfig	Client	Client → Server	JSON-RPC	Tell agent "send webhooks to X".	Payload → {url, token?, authentication.schemes[]}
TaskPushNotificationConfig	Server	Server → Client	JSON-RPC result	Echo / confirm webhook details.	Returned structure mirrors client request (*id*, *pushNotificationConfig*).
Agent-side URL Challenge	Server	Server → Notification URL (GET)	HTTPS	Prove URL is reachable & owned by receiver.	• Server appends *validationToken*. • Receiver echoes token → 200 OK.
Notification Signing Options	–	Server → Receiver	HTTPS	Prevent spoofing & tampering.	• Asymmetric (JSON Web Token (JWT) + JSON Web Key Set (JWKS)) — Server signs body, Receiver validates with public key. • Symmetric (Hash-based Message Authentication Code (HMAC)/JWT) — Shared secret. • OAuth Access Token — Server adds *Authorization: Bearer ...*, Receiver introspects. • Opaque Bearer Token — Simpler but no payload integrity.
Replay Protection	Server & Receiver	–	header/body clock	Ensure one-time delivery.	Include *iat* (issued-at) or nonce inside signed token; Receiver rejects if older than ∼5 min or nonce re-used.
Key Rotation	Server Ops	–	JWKS endpoint	Zero-downtime cert roll-over.	• Publish new JWK alongside old. • Receiver trusts both for overlap period.

exchange real-time *TaskstatusUpdateEvents* and *TaskArtifactUpdateEvents* as the Task progresses. In this "connected" mode, multiple *Tasks* can be multiplexed over the same SSE connection, reducing overhead and latency. Clients send subsequent *Tasks/send* calls on the same Task ID to supply new information or answer follow-up questions, while agents push incremental updates without requiring explicit polling. If the client temporarily loses connectivity, it can reattach to the stream using the *Tasks/resubscribe* method, picking up precisely where it left off and avoiding any gaps in the notification timeline.

- Disconnected Notifications
  Not all clients remain online continuously. For workflows spanning extended durations—minutes, hours, or days—the protocol's push-notification facility ensures the client is informed when a Task reaches a key milestone. When the Task transitions to a terminal or user-input state (such as completed, failed, or input-required), the agent dispatches the full Task payload via an HTTP request to a *preconfigured* notification endpoint. This decouples progress reporting from the client's connection status: even if the client is offline, it can receive updates through an external *NotificationService*, which acts as a secure relay into the client's ecosystem.
- Setting Task Notifications
  To leverage asynchronous callbacks, clients include a *PushNotificationConfig* either in their initial *Tasks/send* invocation or via the *Tasks/pushNotification/set* RPC. This config specifies: (i) url: the callback endpoint for *Task* update POSTs, (ii) token (optional): a one-time or per-*Task* bearer string, and (iii) an authentication block listing acceptable schemes (e.g. ["jwt"][5]). The agent stores this configuration alongside the *Task* and uses it to trigger notifications at appropriate state changes. By bundling this setup into the *Task* creation call, clients streamline orchestration, avoiding separate handshakes or manual subscription flows.
- Agent Security
  Because a malicious actor could specify an arbitrary callback URL, agents must validate each *PushNotificationConfig* before use. A common pattern is a one-time challenge: the agent issues an HTTP GET to the specified URL, including a random *validationToken* in the query

[5]https://jwt.io/

string or headers. Only if the service echoes that token verbatim in its response will the agent trust the endpoint for subsequent POSTs. This handshake prevents agents from inadvertently participating in denial-of-service attacks. Enterprises may tighten this by requiring mutual TLS or IP allow-listing, ensuring that only authorized *NotificationServices* receive *Task* updates.

- Notification Receiver Security
  On the receiving end, clients must verify the authenticity and integrity of incoming push notifications. A2A supports several mechanisms:
  - Asymmetric Keys
    The agent signs each notification—either the entire *Task* object or a JWT wrapper—using its private key (e.g. Elliptic Curve Digital Signature Algorithm (ECDSA) or Rivest, Shamir, Adleman (RSA)). The receiver fetches the agent's public key via a trusted JWKS endpoint or manual configuration and validates the signature. Asymmetric signatures ensure that only the genuine agent could have generated the notification, and they support key rotation without shared secret distribution.
  - Symmetric Keys
    Alternatively, both the agent and receiver share a secret key used in an HMAC signature or a symmetric-key JWT. The receiver recomputes the HMAC over the payload and compares it to the signature header. This approach is simpler to implement but requires secure, out-of-band secret exchange and careful rotation policies to avoid key compromise.
  - OAuth
    In OAuth-based flows, the agent obtains a short-lived access token from an authorization server and includes it in the HTTP *Authorization: Bearer <token>* header when delivering the notification. The *NotificationService* validates the token's audience, issuer, and scopes by introspecting or verifying its signature. This method leverages existing enterprise identity infrastructure and allows centralized revocation of notification privileges.
  - Bearer Token
    A more basic form of webhook authentication is a static bearer token, pre-shared between agent and receiver. Each notification request carries Authorization: Bearer <static-token>, which the receiver matches against its whitelist. While straightforward, static tokens risk leakage if transmitted in plaintext, so they should be rotated frequently or replaced with more robust schemes once stability is achieved.

  Across all these patterns, additional safeguards—such as embedding an issued-at (iat) timestamp, enforcing a narrow time window for acceptance, and tracking jti (i.e. JWT ID) values to prevent replays—ensure that notifications cannot be captured and replayed by adversaries. By combining careful endpoint validation, strong cryptographic authentication and replay controls, A2A's push-notification subsystem guarantees that clients receive only genuine, timely updates on their long-running *Tasks*, regardless of network connectivity.
- Replay and Key Rotation
  Replay prevention is paramount in scenarios where notifications may be intercepted or replayed. Agents embed a timestamp or nonce in each signed or tokenized callback. Recipients track recently seen nonces or enforce a narrow time window. Coupled with sequence numbers in event metadata, this approach thwarts repeated or out-of-order deliveries, ensuring clients only act once per unique *Task* state transition. Key rotation further enhances security. By publishing the agent's public keys via a JWKS[6] endpoint, clients can automatically fetch new keys and retire old ones without downtime. When an agent rotates its signing keys, it adds the new key to the JWKS document alongside the old key, enabling continuous validation of incoming notifications during the transition period. Recipients cache the JWKS document with short Time to Lives (TTLs), periodically refreshing to pick up changes and maintain a seamless trust fabric. Table VII shows key details about the core objects and their life cycle.

D. A2A Communication

Agent-to-Agent Communication revolves around a choreographed exchange between two distinct roles—a "Client" agent that originates a user's request and a "Remote" agent that fulfills it—framed entirely as the life cycle of a *Task* object. When a Client wants something done, it packages the user's intention into a *Task* message and sends it to the Remote agent's HTTP endpoint using JSON-RPC. That initial call (typically via the '*Tasks/send*' method) creates a *Task* with a unique identifier and a status of "submitted."

From there, the Remote agent assumes responsibility for driving that *Task* to completion. For simple requests—say, "translate this sentence"—the agent may process the instruction synchronously and immediately return a *Task* response whose status flips to "completed" and includes one or more *Artifacts* (the translated text, in this case). More often, however, the work is long-running—retrieving data from multiple systems, consulting external APIs, or involving human review. In those scenarios, the Remote agent changes the *Task* status to "working" and continues processing in the background. To keep the Client informed during extended operations, A2A supports two complementary mechanisms. First, the Client can periodically poll the Remote agent by invoking '*Tasks/get*' with the Task ID; the agent responds with the latest *Taskstatus*, any newly available *Artifacts*, and an updated timestamp. Polling works well in environments where persistent connections are impractical or where the Client simply prefers synchronous fetches.

Second, for real-time, push-style updates, the protocol leverages SSE. If both sides advertise "streaming" capability in their *Agent Cards*, the Client may call '*Tasks/sendSubscribe*' instead of '*Tasks/send*'. This call establishes an HTTP connection that the Remote agent holds open,

[6]https://datatracker.ietf.org/doc/html/rfc7517

TABLE VII: Life Cycle of Core Objects in A2A Protocol

Object	**Created / First Owned By**	**Mutated By**	**Primary Hop(s) (Transport & Locality)**	**Wire Direction**
Task	Client (generates *id*, optional *sessionId*; embeds an initial Message)	Server (sets/updates *status*, appends *Artifact[]*, log *history[]*)	• JSON-RPC *Tasks/send* (Client → Server) — carries *id/sessionId/message* in *TasksendParams* • JSON-RPC result or SSE event or Webhook POST (Server → Client) — returns the enriched Task snapshot	Initially Client → Server (skeleton), then Server → Client (full snapshot)
Artifact	Server	Server (may stream additional *Parts*, set *append/ lastChunk*)	• Embedded inside SSE *TaskArtifactUpdateEvent* or final RPC/Webhook Task • Always Server → Client	Uni (Server → Client)
Message	Either actor	Both actors (each new turn)	• Sent inside *TasksendParams* (Client → Server) • Reflected back in histories, status, or input-required prompts (Server → Client)	Bi-directional
Part (child of Message or Artifact)	Same creator as its parent (Client or Server)	Immutable once sent	Carried inside JSON bodies/SSE payloads	Same as parent
PushNotificationConfig (plus returned TaskPushNotificationConfig)	Client embeds in *TasksendParams* or calls *Tasks/ pushNotification/ set*	Server only echoes/stores; never changes content	• JSON-RPC param (Client → Server) • Confirmation RPC result (Server → Client) • Server later POSTs full Task to *url*	• Uni (Client → Server) for config • Uni (Server → Client) for confirmation & webhook POST

streaming back *TaskstatusUpdateEvents* (state transitions like "input-required," "working," or "completed") and *TaskArtifactUpdateEvents* (new or appended content *Parts*) as soon as they occur. SSE dramatically reduces latency and avoids the overhead of repeated polling, making it ideal for highly interactive or time-sensitive workflows. Even SSE doesn't cover every use case. When Clients are offline or when Task durations span hours or days, push notifications provide a fallback. By supplying a *PushNotificationConfig*—essentially a webhook URL and optional authentication tokens—Clients instruct Remote agents to POST the full Task object to that callback endpoint whenever the Task reaches a terminal status or requires manual intervention. This decouples update delivery from any single network session while ensuring Clients won't miss critical state changes.

Underneath these delivery mechanics, the heart of communication is the Task object itself. Each Task encapsulates its current status (one of "submitted," "working," "input-required," "completed," "canceled," "failed," or "unknown"), an optional human- or agent-generated Message explaining the status, and a collection of *Artifacts* containing result data. *Messages* and *Artifacts* are both composed of typed *Parts*—*TextParts*, *FileParts*, or *DataParts*—allowing agents to negotiate rich modalities like HTML views, binary attachments, or structured JSON payloads. If a Task ever moves into "input-required," the Remote agent can include a Message that spells out what credentials or additional context it needs. The Client then obtains those materials out-of-band—perhaps by triggering an OAuth flow—and resubmits them in a follow-up '*Tasks/send*' call, resuming the same Task session.

E. A2A Architecture

Before the A2A Client can dispatch any work, it must first discover the metadata—known as an *Agent Card*—for each remote agent it might call. In our example there are three discovery paths in parallel. For the Open (Direct) Discovery of Calculator Agent A, the client performs a DNS query (UDP/TCP 53) for '*calc.example.com*', receives its IP (i.e.*203.0.113.10*), then issues an HTTPS GET to '*https://203.0.113.10/.well-known/agent.json*' and retrieves *Agent Card*(A), which declares its HTTP endpoint, JSON-RPC capabilities (no streaming, no push notifications), authentication schemes, and the "multiply" skill. Simultaneously, for Registry-Based Discovery of Explainer Agent B and Formatter Agent D, the client resolves '*registry.company.com*' via corporate DNS (i.e. *10.20.0.15*), calls the Enterprise Registry API ('*GET /agents/catalog*') to obtain a vetted list of URLs, and then issues two HTTPS GETs—one to '*https://exp.reg/agent.json*' (yielding *Agent Card* (B), which supports SSE streaming for the "explain" skill) and another to '*https://fmt.reg/agent.json*' (*Agent Card*(D), which offers synchronous formatting into Markdown). Finally, in a locked-down enterprise zone, for Private-API Discovery of Logger Agent C the client resolves 'internal.api.company' (i.e. *10.30.5.7*) and performs an authenticated HTTPS GET against the Private-API Gateway ('*/agents/logger*'), retrieving *Agent Card*(C) that advertises push-notification support for its "audit" and "log" skills.

Once all *Agent Cards* are in hand, the client proceeds to task initiation. The user's original request—"Compute 5×7, explain, pretty-print, audit."—arrived over a standard HTTPS REST call. The client now issues four JSON-RPC calls over HTTPS/TCP 443:

- A synchronous '*Tasks/send*' to Calculator Agent A (Task A1 with a "user" Message containing "5×7"). Agent A immediately returns HTTP 200 with a completed Task object and an *Artifact Part* containing the text "35."
- A '*Tasks/sendSubscribe*' to Explainer Agent B (Task B1 referencing the prior Artifact) over HTTP, initiating

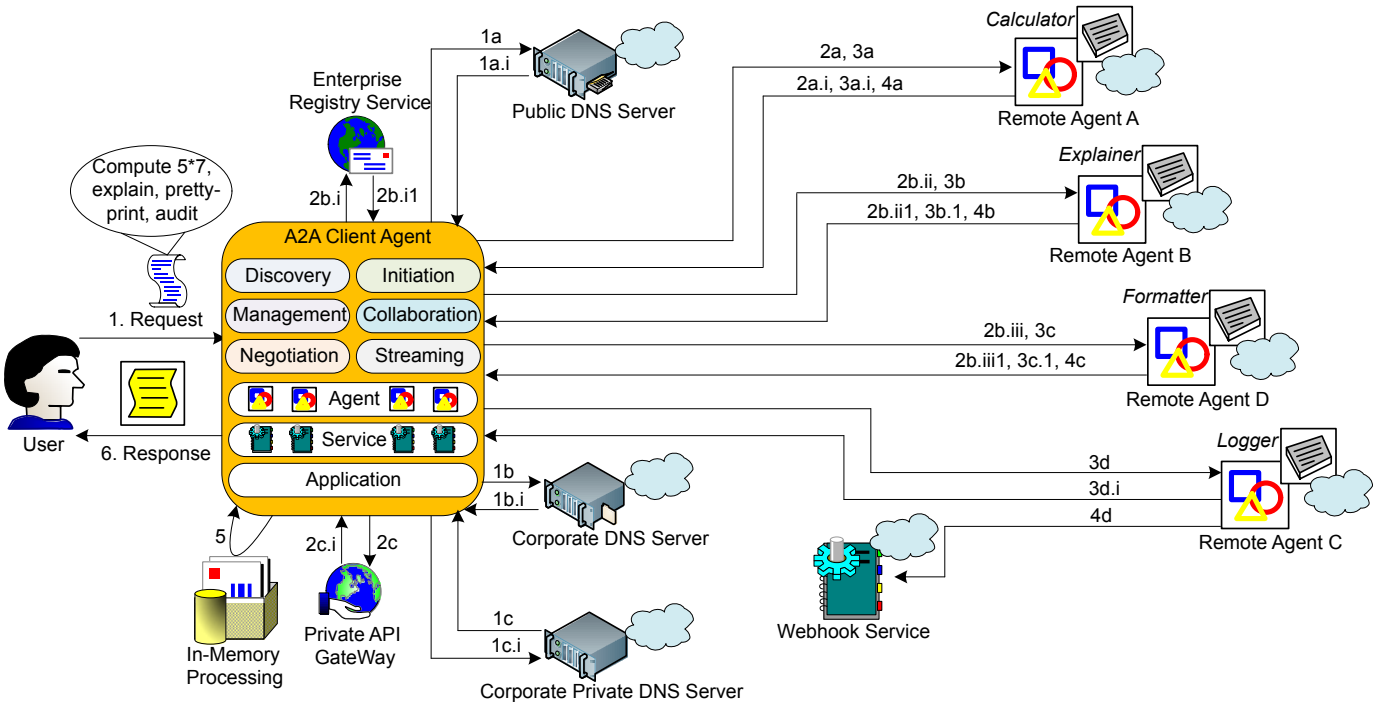

Fig. 2: A2A architecture with various actors and core objects

an SSE stream. Agent B emits a series of '*TaskstatusUpdateEvent*' and '*TaskArtifactUpdateEvent*' frames describing and chunking its natural-language explanation.

- A synchronous '*Tasks/send*' to Formatter Agent D (Task D1 with the explanation text), which replies with a completed Task and a Markdown Artifact in its HTTP response.
- A synchronous '*Tasks/send*' to Logger Agent C (Task C1, including a *PushNotificationConfig* that points at the client's webhook URL), which returns HTTP 202 Accepted to acknowledge receipt.

At last, the result delivery unfolds along three parallel paths: Calculator Agent A's HTTP response delivers the "35" *Artifact*; Explainer Agent B's SSE stream continuously pushes explanation chunks; Formatter Agent D's HTTP response returns the Markdown-formatted text; and Logger Agent C, upon finishing its audit, issues an outbound HTTPS POST to the specified webhook URL carrying the completed *Task* object (status = "completed"), to which the webhook responder returns HTTP 200 OK. Having gathered all *Artifacts* and *Task* updates in memory, the A2A Client merges them into a single coherent result and returns it to the user over HTTPS. This end-to-end flow cleanly separates discovery (e.g. DNS, public well-known paths, registry APIs, private-API gateways) from task orchestration (JSON-RPC over HTTP/S, SSE streaming, webhook notifications) and the core A2A data models (*Agent Card*, *Task, Message, Artifact*, *TaskstatusUpdateEvent*, *PushNotificationConfig*). The Figure 2 presents overall A2A architecture. The abbreviations of data lines in Figure 2 are explained in Table VIII.

F. Intersection with MCP

Modern enterprise AI deployments often require both tightly controlled tool access for individual language model sessions and broad, long-running coordination among multiple autonomous agents [45], [46]. The MCP and the A2A address these needs at complementary layers of the stack. MCP is designed to give a single LLM "inside-the-model" access to external resources—prompts, data stores, and APIs—while enforcing strict isolation and schema validation. When an MCP client process initializes, it negotiates a JSON-schema manifest with its host, learning exactly which "roots" (prompt contexts) and "tools" (i.e. APIs with typed input/output signatures) it may call [47]. Thereafter, the client and server exchange JSON-RPC *Messages* over HTTP, WebSockets, or even stdio for local plugins. The model can then fetch new prompt roots, invoke tools with guaranteed valid arguments, stream partial results (e.g. progress logs via SSE), and politely cancel or renegotiate capabilities if, for example, user consent changes. All authentication—OAuth tokens, API keys, or mTLS certificates—remains out of sight of the model, with the host enforcing policies and attaching credentials at transport time.

By contrast, A2A is built for orchestrating independent, opaque agents—each thought of as a "black box" service—into complex workflows. Every A2A agent publishes an *Agent Card* at a well-known URL (or via a registry or private API) that lists its identity, skills, supported content types, authentication schemes, and streaming or notification capabilities. A client agent discovers candidates either by directly resolving DNS → '*/.well-known/agent.json*', querying an enterprise-curated registry catalog, or invoking a private, authenticated API. Once the appropriate *Agent Card* is re-

TABLE VIII: End-to-End A2A Flow With Transports

Step	Sender → Receiver	Purpose / Action	Outbound Transport / Layer	A2A Payload Object(s)	Return Transport / Layer	Network Response / Payload
A. User Request						
1	User → Client	"Compute 5×7, explain, pretty-print, audit."	HTTPS REST (non-A2A)	–	Same HTTPS TCP 443	HTTP 200 + JSON UI payload
B. Open (Direct) Discovery — Calculator A						
1a	Client → Public DNS	Resolve *calc.example.com*	UDP/TCP 53 DNS Query	–	1a.i. UDP/TCP 53 DNS Response	A/AAAA = *203.0.113.10*
2a	Client → *calc.example.com*	Download AgentCard(A)	HTTPS GET /.well-known/agent.json	–	2a.i. HTTPS TCP 443	HTTP 200 + AgentCard(A)
C. Registry Discovery — Explainer B, Formatter D						
1b	Client → Corp DNS	Resolve *registry.company.com*	DNS A/AAAA	–	1b.i. DNS Response	*10.20.0.15*
2b.i	Client → Enterprise Registry	Fetch vetted list	HTTPS GET /agents/catalog	–	2b.i1. HTTPS	HTTP 200 + list of URLs
2b.ii	Client → exp.reg/agent.json	Card for Explainer B	HTTPS GET	–	2b.ii1. HTTPS	HTTP 200 + AgentCard(B)
2b.iii	Client → fmt.reg/agent.json	Card for Formatter D	HTTPS GET	–	2b.iii1. HTTPS	HTTP 200 + AgentCard(D)
D. Private-API Discovery — Logger C						
1c	Client → Corp DNS	Resolve *internal.api.company*	DNS A/AAAA	–	1c.i. DNS Response	*10.30.5.7*
2c	Client → Private-API GW	Auth card lookup (mTLS/OAuth)	HTTPS GET /agents/logger	–	2c.i. HTTPS	HTTP 200 + AgentCard(C)
E. Task Dispatch						
3a	Client → Calculator A	Sync multiply 5×7	HTTP POST JSON-RPC *Tasks/send*	Task(A1), Message(text)	3a.i. Same HTTP TCP 443	HTTP 200 + Task(completed), Artifact(text=35)
3b	Client → Explainer B	Stream explanation	HTTP POST JSON-RPC *Tasks/ sendSubscribe*	Task(B1), Message	3b.i. SSE over HTTP 2xx	Continuous *Taskstatus / Artifact* events
3c	Client → Formatter D	Markdown formatting	HTTP POST JSON-RPC *Tasks/send*	Task(D1), Message	3c.i. HTTP TCP 443	HTTP 200 + Task(completed), Artifact(md)
3d	Client → Logger C	Audit + webhook cfg	HTTP POST JSON-RPC *Tasks/send*	Task(C1), Message, PushNotifCfg	3d.i. HTTP TCP 443	HTTP 202 Accepted
F. Result Paths						
4a	Calculator A → Client	Immediate result "35"	(HTTP response of 3a)	Task, Artifact	–	Same response body
4b	Explainer B → Client	Live explanation chunks	SSE	Taskstatus UpdateEvent,TaskArtifact UpdateEvent	SSE text/event-stream	Streamed frames
4c	Formatter D → Client	Markdown page	(HTTP response of 3c)	Task, Artifact(md)	–	Same response body
4d	Logger C → Webhook	Async audit confirmation	HTTPS POST webhook	Task(status completed)	HTTPS	HTTP 200 OK from webhook svc
G. Aggregation & Return						
5	Client (local)	Merge artefacts	In-memory	*Tasks + Artifacts*	–	–
6	Client → User	Render final answer	HTTPS REST UI	–	HTTPS	HTTP 200 JSON/HTML

trieved, a new *Task* object (with unique ID and lifecycle state) is created and sent via JSON-RPC over HTTPS—either as a one-off '*Tasks/send*' or a streaming '*Tasks/sendSubscribe*'. For quick *Tasks*, the remote agent returns a completed Task and its *Artifacts* immediately. For longer operations, it streams '*TaskstatusUpdateEvent*' and '*TaskArtifactUpdateEvent*' *Messages* over SSE, or—when the client is offline—fires a webhook-style *PushNotification* to a preconfigured URL. Should the agent require further input (e.g. credentials, parameter clarifications), it returns the *Task* in an '*INPUT-REQUIRED*' state along with an Authentication schema, prompting the client to handle the out-of-band credential exchange before resuming. Once all participants finish, the client aggregates the resulting *Artifacts*—text, images, structured data—and presents a unified response back to the user.

While both MCP and A2A rely on standard HTTP security (e.g. OAuth bearer tokens, API keys, mTLS), they expose credentials differently. MCP hides all authentication details behind the host application, keeping the model's view strictly to typed tool signatures. A2A surfaces each agent's supported schemes directly in its *Agent Card* and only asks for new credentials mid-task if required, preventing any mixing of sensitive tokens with business payloads. Likewise, MCP's strict JSON-Schema validation ensures compile-time safety for well-known APIs, at the cost of rigidity, whereas A2A's "*Parts*" model—text, file, or data with loose MIME-style metadata—trades off schema strictness for the flexibility needed in multi-modal, evolving agent workflows. Finally, observability and error handling span all layers: MCP tools return structured

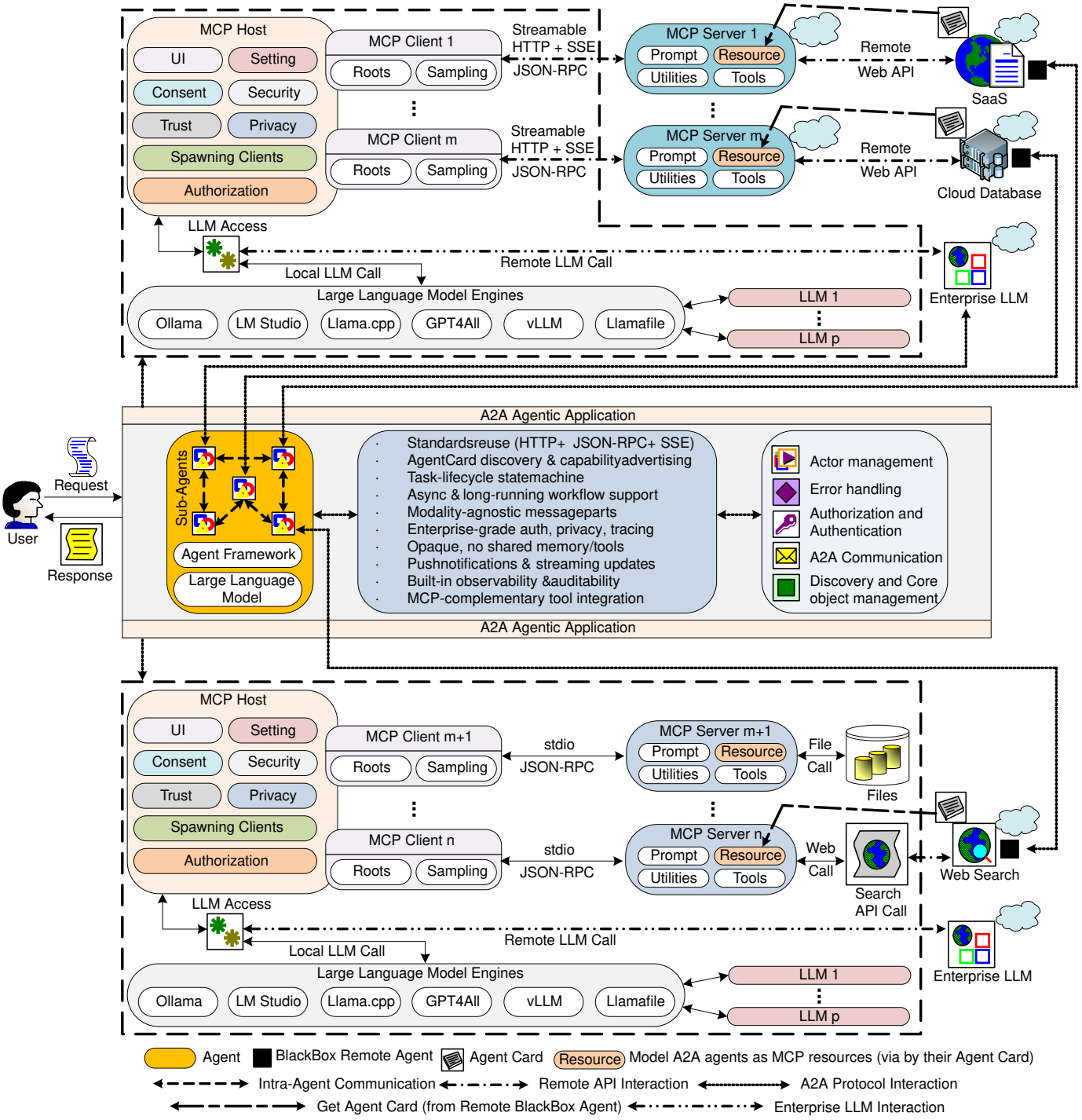

Fig. 3: A2A and MCP interaction architecture

error objects, A2A *Tasks* include explicit 'failed' states and human-readable failure *Messages*, and HTTP status codes continue to anchor traditional API interactions. Together, MCP and A2A form a layered, enterprise-grade architecture in which APIs deliver raw services, MCP transforms them into model-centric tools, and A2A orchestrates agent-level collaboration—unlocking dynamic, federated AI automation across organizational boundaries. A comparative overview of MCP and A2A with respect to traditional APIs is shown in Table IX.

The Figure 3 illustrates how the MCP and A2A protocols interoperate within a layered AI-agent architecture. At the top and bottom, dashed "MCP Host" boxes show a host process managing multiple MCP clients—each encapsulating Roots and Sampling managers—and negotiating capabilities over JSON-RPC, either via HTTP+SSE for remote services or *stdio* for local subprocesses. These clients connect to their corresponding MCP servers, which expose modular services (e.g. *Prompts, Resources, Tools, and Utilities*) backed by remote Web APIs, cloud databases, or SaaS endpoints. Between these

TABLE IX: Comparative Overview of A2A, MCP, and Traditional APIs

Aspect	**Agent-to-Agent**	**Model Context Protoco)**	**Traditional API**
Primary Purpose	Peer-to-peer task delegation and coordination	Enriching LLM inference with external tools and data during execution	Exposing create, read, update and delete (CRUD) or RPC endpoints for traditional applications
Discovery Mechanism	*Agent Cards* via well-known URL path	Pre-configured registry or manifest on client-side	API catalogs (e.g. Swagger[7]/OpenAPI specifications)
Communication Model	JSON-RPC over HTTP(S) with optional SSE or webhooks	Synchronous RPC during a single inference session	Request/Response over HTTP(S) or gRPC
State Management	Stateful task lifecycle (submitted, working, completed)	Stateless tool calls; context maintained by the host	Typically stateless per request; sessions optional
Modality Support	Multi-part *Messages* (e.g. 7text, images, videos, forms)	JSON-structured inputs/outputs, binary blobs	Primarily JSON or XML payloads
Security Model	OAuth2, mTLS, API Key declared in Agent Card; enforced per request	Host-defined (OAuth2 tokens, custom headers)	API Key, OAuth2, Basic Authentication at transport layer
Tool/Service Integration	Agents invoke each other's *Tasks* as opaque services (Agent-as-Service model)	LLMs invoke tools via predefined schemas (Tool-as-Service model)	Endpoint-based service invocation with strict payload contracts
Concurrency	Supports concurrent long-running *Tasks*, streaming updates, and push notifications	Single inference session; concurrency needs orchestration at host level	Stateless servers allow easy horizontal scaling
Error Handling & Semantics	JSON-RPC standard error codes, plus task-specific failure states	Inline tool errors; host surfaces them to LLM with reasoning context	HTTP status codes (4xx/5xx); often custom error payloads
Versioning	Declared in Agent Card; clients negotiate schema versions dynamically	Out-of-band API version management; MCP servers handle evolution	URI versioning or HTTP headers negotiated via OpenAPI specifications
Extensibility	Open-ended "*Parts*" model for new content types; flexible Agent Card extensions	JSON Schema-based tool descriptions; easy to add or modify tools	OpenAPI extension mechanisms; versioned models and paths
Performance & Latency	Higher overhead due to RPC handshake and streaming; optimized for bursty workloads	Minimal overhead per call; depends on local/remote server proximity	Tuned via HTTP keep-alive, caching, load balancing
Use Case Focus	Multi-agent orchestration in enterprise systems and agent ecosystems	Augmentation of single LLM sessions with dynamic tool use	General-purpose client-server integration (web, mobile, backend systems)
Ecosystem Alignment	Enterprise AI workflows, decentralized agent networks	LLM tool-calling standardization across applications	Classic software systems, web services, microservices

MCP layers sits the A2A Agentic Application: a collection of sub-agents built on a lightweight framework that embeds a large-language-model engine (e.g. Ollama [48], Llama.cpp [49], GPT4All [50], vLLM [51]) and reuses HTTP+JSON-RPC+SSE for communication. Each agent advertises its capabilities via a well-known *Agent Card*, supports task life cycle management, modality-agnostic message *Parts*, push-notification streaming, enterprise-grade authentication/authorization, and built-in observability. Arrows depict intra-agent RPC calls, remote API interactions, and LLM invocations—local calls to on-premise models or remote calls to enterprise LLM services—while the legend clarifies symbol usage for agents, black-box endpoints, resources, and the mapping of A2A agents as MCP resources. Together, this diagram conveys a unified vision in which MCP provides tightly typed, low-latency tool integration for individual LLM sessions, and A2A orchestrates multi-agent workflows, discovery, and long-running *Tasks* across organizational boundaries.

IV. State-of-the-art

This section explores how the A2A protocol enables dynamic, secure, and modular coordination across a variety of real-world domains. By leveraging core abstractions such as *Agent Cards*, *Tasks*, *Messages*, and *Artifacts*, A2A facilitates the orchestration of interoperable agents capable of exchanging structured information and status updates over standardized transport layers. We examine prospective use cases across enterprise IT, customer support, logistics, healthcare, and education—highlighting how autonomous agents collaborate under the A2A framework to deliver scalable, traceable, and efficient workflows.

A. Prospective Use Cases

This section presents a range of real-world scenarios where an open, metadata-driven agent framework can be applied to automate and coordinate complex processes. We demonstrate how independently developed services—each advertising its capabilities through a common interface—can be dynamically discovered, engaged, and monitored to execute end-to-end workflows with secure handoffs, live progress updates, and callback notifications across domains such as IT operations, customer support, supply chain management, and beyond. In most of the use cases, we envisage an example where A2A can be applied with feasible manner.

1) Enterprise Workflow Automation: In a typical enterprise, complex processes span multiple domains—IT, HR, facilities—and require tight coordination [52]. With A2A, an "ops-orchestrator" agent can discover domain-specific agents via their JSON *Agent Cards*, each advertising skills like "incident-detection" or "hardware-procurement." When a server anomaly is detected, the orchestrator issues a *Task* to the incident-detection agent. That agent streams status updates over SSE—sending *Tasks*statusUpdateEvents indicating "investigating," "root-cause found," etc.—and appends diagnostic logs as Text*Parts* in *Artifact* chunks. Upon confirmation of hardware failure, the orchestrator dispatches a

new *Task* to the procurement agent, including a *DataPart* containing the equipment specification. The procurement agent returns an *Artifact* with an order confirmation PDF *FilePart*. Finally, a maintenance agent receives a push-notification via its PushNotificationConfig callback URL and schedules service. Throughout, each agent transitions *Task.state* (e.g. "working," "input-required," "completed") so the orchestrator has a unified view of progress and can handle exceptions or manual approvals inline.

2) Customer Support Enhancement: Customer support teams often need to tap into billing, account, and technical diagnostics systems simultaneously. Using A2A, a front-line "support-bot" agent may first formulate a *Task* encapsulating the customer's issue—say, "refund overcharge" with a *TextPart* describing the transaction. It queries a "billing-engine" agent, which authenticates via OAuth, processes the refund, and returns a JSON *DataPart* detailing the credit amount. In parallel, if the issue involves a product defect, the support-bot spawns a *Task* to a "diagnostic-agent," streaming back logs and troubleshooting advice as multi-part *Artifacts*. If further input is required—e.g. photos of the defect—the diagnostic agent emits a *Tasks*tatusUpdateEvent with state='input-required' and an instruction Message. The customer's response arrives as a FilePart, and the diagnostic agent resumes. When both billing and technical *Tasks* reach "completed," the support-bot aggregates their *Artifacts* into a final summary, sent back to the user. This modular, real-time collaboration yields faster resolution and consistent audit trails [53].

3) Supply Chain and Logistics Optimization: In global supply chains, maintaining just-in-time inventory requires constant data exchange and predictive analytics. An A2A-enabled "supply-control" agent can first enquire an "inventory-monitor" agent via a *Task* that streams current stock levels as *DataParts*. If threshold breaches occur, "procurement" and "forecasting" agents are tasked concurrently: the forecasting agent ingests sales trends via historical *DataParts*, runs machine-learning models, and returns demand projections as JSON. The procurement agent uses those projections to generate purchase orders, emitting an *Artifact* containing the PO document. Meanwhile, a "shipping-coordination" agent receives the PO *Artifact* and schedules carriers, streaming tracking updates via SSE [54]. If a delay is predicted, it pushes a notification to the supply-control agent, which can initiate a fallback *Task* with alternate carriers. By chaining these *Tasks* and *Artifacts*, A2A furnishes a resilient, automated supply-chain control loop.

4) Healthcare Coordination: Patient care often spans scheduling, record management, and treatment planning, each with stringent privacy and audit requirements [55]. A2A's secure by-default design—leveraging TLS, OAuth scopes, and fine-grained skills may enable a "care-coordinator" agent to interact with discrete "scheduling," "records," and "treatment-planning" agents. First, the coordinator sends an "initiate-visit" *Task* to scheduling, embedding patient demographics as encrypted *DataParts*. Scheduling replies with appointment details as *TextParts*. Next, the coordinator invokes records access; the records agent transitions to 'input-required' to request patient consent, then streams de-identified health history via *DataParts*. Finally, a treatment agent receives the history *Artifact* and returns a care plan PDF as a *FilePart*. All status updates and *Artifacts* are logged in an immutable *Task.history*, satisfying compliance needs while eliminating paper-based handoffs.

5) Financial Services Automation: Banks and fintech firms require robust workflows for loans, fraud detection, and customer inquiries. An A2A-compliant "loan-originator" agent can initiate a "loan-application" *Task* that includes applicant data as a JSON *DataPart*. It calls a "credit-scorer" agent, which authenticates via a shared API key, retrieves credit bureau info, and returns a score as a *DataPart*. If the score is below threshold, a "manual-review" agent gets an SSE stream of flagged criteria [56]. Once approved, a "document-generator" agent compiles loan agreements as a PDF *Artifact*. In parallel, a "fraud-monitor" agent subscribes to transaction updates, issuing push notifications to the loan-originator if anomalies appear. This concurrent, stateful exchange reduces time to decision while ensuring each agent's specialized logic remains decoupled yet fully auditable.

6) E-Commerce Personalization Orchestration: Delivering tailored shopping experiences demands real-time data and coordination among recommendation, pricing, and checkout subsystems [57]. Using A2A, an "order-assistant" agent can package a *Task* with the user's browsing history as *DataParts* and sends it to a "product-recommender" agent. The recommender streams back ranked items via repeated *Artifact* chunks (JSON lists). Upon selection, the assistant dispatches a *Task* to a "promotion-engine" agent, passing the cart contents and user loyalty status; the engine returns discount codes as Text*Parts*. Finally, a "checkout-optimizer" agent is tasked to negotiate shipping options and payment gateways. It may enter 'input-required' to prompt the user for a payment method, then emits an interactive payment form Part. Once the user completes payment, the optimizer returns a confirmation *Artifact*. This end-to-end chain leverages A2A's modality-agnostic *Parts* and stateful *Tasks* to deliver a seamless, highly personalized purchase flow.

7) Recruitment and Talent Acquisition: Hiring pipelines involve multiple specialized steps—sourcing, interviewing, vetting—that can each be handled by dedicated agents. In A2A, a "talent-manager" agent may begin with sending a "source-candidates" *Task* to a sourcing agent, providing job requirements as a *DataPart*. The sourcing agent streams candidate profiles as repeated *Artifact* chunks, each containing structured JSON with skills, experience, and contact info. Once candidate lists are assembled, the manager agent issues an "schedule-interviews" *Task* to a calendar agent, passing available time slots as *DataParts*. The calendar agent negotiates with interviewer and candidate agents via SSE, returning confirmed meeting invites as ICS *FileParts*. After interviews, a "background-check" agent is invoked via push notification; its callback URL delivers verification reports as PDF *FileParts* when complete. At each handoff, *Task.status* and history ensure the talent-manager maintains end-to-end visibility, vastly reducing manual coordination.

8) Smart City Infrastructure Management: Coordinating transport, energy, and utilities in a smart city involves hetero-

geneous Internet of Things (IoT) data and real-time control loops [58], [59]. A "city-ops" agent may use A2A to interact with domain agents: it opens a *Task* with the "traffic-monitor" agent, which streams congestion maps as *ImageParts* over SSE. Simultaneously, it *Tasks* an "energy-balancer" agent with grid load data as *DataParts*; the balancer streams back control commands to adjust transformer settings. If the water usage spikes, a "utilities-agent" receives a push notification containing the event and responds with pressure-adjustment instructions as *DataParts*. At any point, if an emergency arises, the city-ops agent switches the *Task* to 'input-required' to secure manual override credentials. By binding all subsystems under one *Task* life cycle, A2A enables synchronized, cross-domain responses to dynamic urban conditions.

9) Collaborative Scientific Research Workflows: Large-scale experiments often entail data collection, preprocessing, modeling, analysis, and publication. In an A2A setting, a "research-coordinator" agent can issue a "collect-data" *Task* to a "data-acquisition" agent, passing instrument configurations as *DataParts*. The acquisition agent streams raw datasets as *FilePart* chunks. Next, the coordinator calls a "data-curation" agent to clean and normalize the data—this agent emits progress *Messages* and appends cleaned outputs as new *FileParts*. For compute-intensive model training, a "model-trainer" agent receives the cleaned data *Artifact* and reports epoch checkpoints via SSE as *DataParts*. Once training completes, a "statistical-analysis" agent performs tests and returns structured results (JSON *DataParts*). Finally, a "publication-generator" agent assembles figures and narrative into a LaTeX PDF *Artifact*. Each agent's capabilities are discovered via *Agent Cards*, and the entire workflow is tracked through a single *Task.history* for reproducibility and audit.

10) Legal Contract Lifecycle Management: Managing contracts involves clause extraction, negotiation, approval workflows, and secure archival [60]. An "intake-agent" may first create a *Task* with the incoming draft as a PDF *FilePart*. A "clause-extractor" agent is discovered via its Agent Card skill "extractClauses," and receives the Task; it streams back extracted clauses as JSON *DataParts*, one per clause. The intake-agent then spawns parallel *Tasks*: a "risk-analyzer" agent ingests the clause *DataParts* and returns a summarized risk report as a *TextPart Artifact*, while a "negotiation-assistant" agent generates proposed redlines as a Word doc *FilePart*. If human review is needed, either agent transitions to input-required with an instruction Message. Once finalized, a "signing-agent" is notified via PushNotificationConfig to deliver e-signature requests, and upon signature completion emits a signed PDF *Artifact*. Finally, an "archive-agent" receives the signed document *Task* and stores it in a document management system, confirming via an SSE status update. Every step is traceable via the *Task.history* for audit and compliance.

11) Personalized Tutoring Networks: In adaptive learning platforms, dynamic content generation and student assessment can be handled by specialist agents. A "course-manager" agent can assign a "lesson-generator" *Task* specifying topic, difficulty level, and student profile in a *DataPart*. The lesson-generator streams back module content as a sequence of *TextPart* and *ImagePart Artifacts* (e.g. slides, diagrams). Concurrently, an "assessment-agent" is tasked to create quizzes: it sends Questions as JSON *DataParts*. After the student completes quizzes, the assessment-agent signals input-required and requests student answers as repeated *DataParts*. The tutor-bot then analyzes responses, streams feedback via SSE, and generates a personalized remediation plan as a PDF *FilePart*. If progress stalls, a push notification alerts a human instructor agent to intervene. By chaining these agents under one *Task*, the platform delivers a fully automated, adaptive learning journey tailored to each student.

12) Disaster Management: Coordinating multi-agency disaster response requires real-time situational awareness and rapid delegation [61], [62]. An "incident-commander" agent should ingest incoming sensor feeds (e.g. flood levels, seismic data) via *FilePart* streams and issues a "damage-assessment" *Task* to a geospatial-analysis agent. That agent returns heat-maps as ImagePart *Artifacts*. Simultaneously, a "resource-allocator" agent receives *DataParts* describing available rescue teams and equipment; it streams back deployment plans as JSON. If road closures occur, the resource-allocator emits a input-required Message prompting the commander to approve rerouting. Once plans are locked in, a "notification-agent" uses *PushNotificationConfig* to broadcast alerts to citizen-facing systems. Finally, a "logistics-agent" subscribes via SSE for updates on supply deliveries and streams back Estimated Time of Arrival (ETA) updates. This tightly orchestrated, stateful flow ensures that multiple specialized agents collaborate seamlessly under the A2A protocol, accelerating response times in critical scenarios.

13) Smart Manufacturing Orchestration: Modern factories rely on coordinated workflows between quality control, maintenance, and production planning systems [63], [64]. A "production-planner" agent can begin by sending a "schedule-run" *Task* to a manufacturing-execution agent, passing batch specifications as *DataParts*. The execution agent streams back machine status updates via SSE—reporting working, temperature logs as *TextParts*, and eventual run completion. In parallel, a "quality-control" agent receives in-line sensor data as *DataParts* and outputs inspection reports as PDF *Artifacts*. If defects exceed thresholds, the quality-control agent transitions to input-required to request manual override instructions. Upon override approval, a "maintenance-agent" is notified via Push Notification and schedules preventive maintenance, updating its own *Task* state back to the planner. Once maintenance completes, the planner automatically reissues the "schedule-run" *Task* to resume production. By linking these agents through A2A's *Task* life cycles, manufacturers achieve dynamic, closed-loop optimization without custom point-to-point integrations.

B. Enterprise Readiness of the A2A Protocol

Supporting robust, secure interoperability among autonomous agents in large organizations demands more than just a message format. The A2A protocol is architected to slot seamlessly into existing enterprise security and operations landscapes. Rather than inventing new cryptographic

or identity schemes, A2A treats every agent endpoint as a conventional HTTP service. By building atop HTTPS, leveraging OAuth/OpenAPI authentication, and integrating with standard observability toolchains, A2A ensures that enterprises can adopt agent-to-agent workflows without rewriting firewall rules, identity policies, or logging infrastructure.

1) Transport Level Security: At its heart, A2A relies on TLS-secured HTTP for all communication. Every A2A call—whether it's *Tasks*/send, *Tasks*/get, or an SSE stream—must traverse an encrypted channel. Production deployments should mandate TLS 1.2+ (ideally TLS 1.3 [65]), disable legacy ciphers [66], and enforce forward secrecy suites (ECDHE) [67]. In practice, organizations often terminate TLS at an API gateway or load balancer (e.g. Envoy[8], NGINX[9], AWS ALB[10]), which centralizes certificate management, istributed Denial-of-Service (DDoS) mitigation, and Web Application Firewall (WAF) policies. Certificates themselves can be provisioned and rotated via Automatic Certificate Management Environment (ACME)-compatible tooling, with automated alerts for impending expiry. For the most sensitive environments, mutual TLS can be layered on top, forcing both client and server to present valid X.509 certificates [68] bound to enterprise trust anchors. This two-way handshake not only encrypts traffic but also authenticates agents before any A2A logic executes.

2) Server Identity: Verifying an agent's true hostname and certificate is the first step in preventing impersonation or man-in-the-middle attacks. During the TLS handshake, a Remote Agent presents an X.509 certificate chain ending in a trusted root CA. A2A clients must perform full chain validation: checking that each intermediate and root CA is trusted, verifying that the certificate is not expired, and consulting Online Certificate Status Protocol (OCSP) [69] or Certificate Revocation List (CRL) [70] endpoints to ensure the certificate has not been revoked. Enterprises can integrate A2A servers into existing certificate management frameworks—whether that's a corporate Public Key Infrastructure (PKI), public issuers, or DNS-based Authentication of Named Entities (DANE) [71] via Domain Name System Security Extensions (DNSSEC) [72]. For extra assurance, clients may employ certificate pinning, recording the expected public key fingerprint during onboarding and rejecting any certificate not matching that pin. This reuse of standard PKI practices means there's no need for bespoke trust stores or out-of-band key exchanges.

3) Client and User Identity: Rather than embedding user IDs or client IDs deep inside the A2A payload, the protocol pushes identity concerns entirely onto the HTTP layer. Every A2A request must carry authentication credentials in headers—be that an OAuth2 Bearer token, an API key, a JWT, or even an mTLS client certificate. This design keeps the JSON-RPC task schemas clean and permits agents to adopt whatever identity scheme suits the enterprise. Clients negotiate these credentials out-of-band with corporate identity providers (IdPs)[11] such as Azure AD[12], Okta[13], or an internal SSO solution. They handle token acquisition, caching, and refresh flows transparently, then inject the resulting tokens into each A2A call. Complex scenarios—where one agent (Agent A) authenticates via the enterprise directory and then needs to call another agent (i.e. Agent B) secured by a third-party SaaS identity—are handled through A2A's "input-required" task status. If Agent A demands additional credentials mid-task, it transitions the *Task* into an INPUT-REQUIRED state and returns an authentication structure describing what it expects. The client then performs the external IdP dance for Agent B (e.g. OAuth code flow), obtains fresh tokens, and resumes the same *Task* by supplying those tokens in a follow-up message. This preserves a secure chain of custody and avoids baking any particular IdP logic into the A2A spec itself.

4) Authenticating Clients: Before invoking any task logic, A2A servers must authenticate every incoming request at the HTTP layer. Each Remote Agent's Agent Card advertises the supported schemes—whether OAuth2 authorization code, client credentials, API keys, HTTP Basic, or custom JWT assertions—in an OpenAPI-style structure. Clients can retrieve this Agent Card from */.well-known/agent.json*, discover the desired scheme, and configure their requests accordingly. During runtime, if a request arrives without valid credentials, the server immediately returns a 401 Unauthorized status, including a WWW-Authenticate header pointing to the required scheme or OIDC discovery endpoint. If credentials are valid but lack the necessary privileges, a 403 Forbidden is returned. This model leverages existing OAuth/OIDC middleware and libraries, so client developers can plug A2A calls into their corporate authentication workflows without custom code. Behind the scenes, opaque tokens can be validated via token introspection endpoints, JWTs can be verified via public JWK sets, and API keys can be looked up in vaults. All of these mechanisms remain outside the formal A2A payload, preserving protocol simplicity while upholding enterprise authentication standards.

5) Authorization and Data Privacy: Authentication answers "who," but authorization governs "what" an identity can do. A2A servers should enforce a two-axis authorization model:

- Skill-Level Authorization
  Each agent advertises its capabilities as named "skills" in the *Agent Card*—e.g. *createInvoice, queryPayroll*. Enterprises assign OAuth2 scopes or IAM roles that map directly to those skills. A token bearing the invoice:create scope can invoke that skill but will be denied if it tries to run, say, *accessEmployeeRecords*. This approach allows administrators to use familiar Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC) controls at the agent boundary.
- Tool-Level Authorization
  Some agents orchestrate calls to backend tools or data connectors (e.g. databases, third-party APIs). Access to

[8]https://www.envoyproxy.io/
[9]https://nginx.org/
[10]https://aws.amazon.com/elasticloadbalancing/application-load-balancer/
[11]https://help.sap.com/docs/identity-authentication/identity-authentication/corporate-identity-providers
[12]https://www.microsoft.com/en-in/security/business/identity-access/microsoft-entra-id
[13]https://www.okta.com/

those sensitive operations should be gated by separate credentials or permissions. For instance, an agent might require an additional API key to modify production databases; before invoking that tool, the agent verifies that the caller's identity possesses a corresponding privilege in a centralized policy engine (e.g. Open Policy Agent[14]). This layered gating ensures that even if an agent itself is trusted, its sub-operations remain under strict control.

In both cases, data minimization should be enforced: agents only request the minimum set of user attributes and data necessary to complete the task. Personally Identifiable Information (PII) and regulated data must be tagged and routed through data classification pipelines, ensuring compliance with General Data Protection Regulation (GDPR)[15], ealth Insurance Portability and Accountability Act (HIPAA)[16], or other regulations. *Artifact* storage—whether in-memory caches or long-term persistence—must be encrypted at rest (i.e. AES-256) and governed by enterprise key management systems (KMS[17], Vault[18]).

6) Tracing and Observability: Visibility into multi-agent workflows is essential for diagnosing latencies, error bursts, and security incidents. Since A2A rides over HTTP, it can adopt OpenTelemetry[19] or W3C Trace Context[20] standards for distributed tracing. Clients and servers propagate a traceparent header on each call; spans are created for RPC methods like *Tasks/send*, *Tasks/get*, *Tasks/sendSubscribe*, as well as for outbound webhooks or database accesses. This end-to-end trace allows Site Reliability Engineering (SRE) teams to visualize the entire lifecycle of a *Task*—across multiple agent hops—and pinpoint performance bottlenecks or unexpected failures. Simultaneously, structured logging should capture key events:(i) *Task* creation, (ii) status transitions (e.g. working, input-required, completed), (iii) authentication failures, and (iv) *Artifact* generation. Logs include metadata fields such as *Task* ID, Session ID, client principal, and skill name. These logs feed into centralized Elasticsearch, Logstash, and Kibana (ELK) stacks[21], Splunk[22], or cloud log analytics [73], enabling security and compliance teams to run audits, generate SLA reports, and set alerts on anomalous patterns (e.g. spike in failed states or unauthorized access attempts). Health endpoints (/healthz, /readyz) integrate with orchestration platforms (e.g. Kubernetes[23], ECS[24]) to drive automated scaling and failover. Metrics—such as per-skill invocation counts, average *Task* latency, and SSE connection counts—are published to monitoring systems (e.g. Prometheus[25], Datadog[26]), allowing proactive capacity planning and Service Level Object (SLO) [74] enforcement.

C. Existing Industry Involvement

The A2A protocol has rapidly attracted contributors from a remarkably broad cross-section of industries. At the core, leading enterprise-software vendors such as Atlassian[27], Salesforce[28], ServiceNow[29], and SAP[30] are embedding A2A into their platforms to stitch together formerly siloed workflows. At Atlassian, for example, A2A–enabled "sub-agents" inside Jira and Confluence can now autonomously coordinate code-review, test-automation, and release-approval *Tasks*—eliminating manual handoffs between development, QA, and operations teams. Salesforce is likewise building A2A into its Agentforce chatbot, allowing marketing, sales, and service bots to delegate leads, quotes, and support cases to one another without custom APIs.

In the data-and-observability realm, partners like Datadog, Elastic[31], Chronosphere[32], DataStax[33], and Neo4j[34] are pioneering "agent meshes" that monitor, analyze, and remediate system health in real time. A Datadog-driven diagnostic agent can detect anomalous metrics, spin up a resource-allocator agent to quarantine problematic nodes, then notify a compliance-agent to generate an audit report—all through standard A2A *Tasks*, streamed status updates via SSE, and completion *Artifacts* for post-mortem analysis. Elastic and Chronosphere are exploring similar multi-agent patterns for tracing, logging, and predictive scaling across hybrid clouds.

Global consultancies—including Accenture, Deloitte[35], Capgemini[36], Cognizant[37], KPMG[38], PwC[39], and McKinsey[40]—are among the most enthusiastic early adopters, embedding A2A best practices into enterprise transformation engagements. These firms are helping Fortune 500 clients decompose monolithic business processes into networks of specialized, reusable agents: HR agents for recruiting and onboarding, procurement agents for vendor management, legal agents for contract review, and more. By codifying each step as a discrete A2A *Task*, clients gain end-to-end traceability, faster iteration on process improvements, and the ability to swap out or upgrade individual agents without refactoring entire workflows.

In financial services, PayPal[41], Intuit[42], S&P Global[43],

[14]https://www.openpolicyagent.org/
[15]https://gdpr-info.eu/
[16]https://www.cdc.gov/phlp/php/resources/health-insurance-portability-and-accountability-act-of-1996-hipaa.html
[17]https://docs.aws.amazon.com/kms/latest/developerguide/overview.html
[18]https://developer.hashicorp.com/vault
[19]https://opentelemetry.io/
[20]https://www.w3.org/TR/trace-context/
[21]https://www.elastic.co/elastic-stack
[22]https://www.splunk.com/
[23]https://kubernetes.io/
[24]https://aws.amazon.com/ecs/
[25]https://prometheus.io/
[26]https://www.datadoghq.com/
[27]https://www.atlassian.com/
[28]https://www.salesforce.com/in/
[29]https://www.servicenow.com/
[30]https://www.sap.com/india/index.html
[31]https://www.elastic.co/
[32]https://chronosphere.io/
[33]https://www.datastax.com/
[34]https://neo4j.com/
[35]https://www2.deloitte.com/us/en.html
[36]https://www.capgemini.com/
[37]https://www.cognizant.com/us/en
[38]https://www.kpmc.in/
[39]https://www.pwc.in/
[40]https://www.mckinsey.com/in/overview
[41]https://www.paypal.com/
[42]https://www.intuit.com/in/
[43]https://www.spglobal.com/en

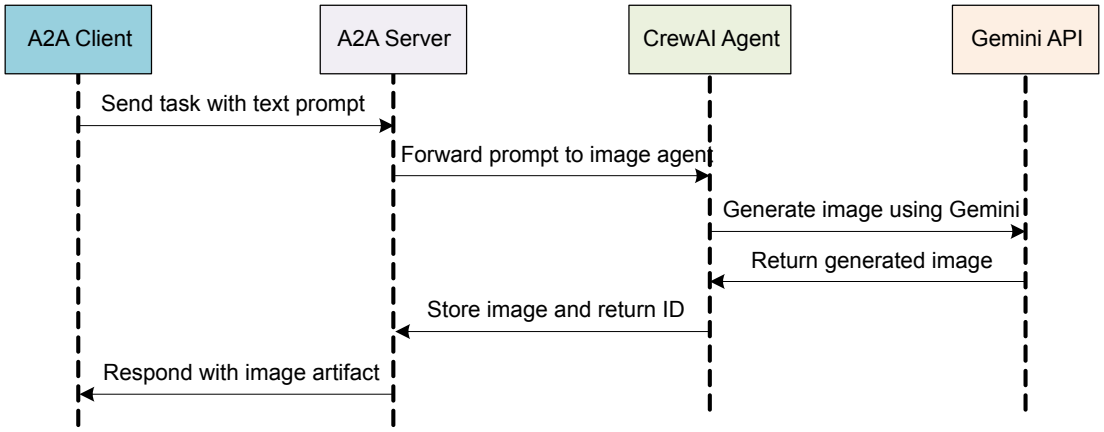

Fig. 4: CrewAI based A2A architecture deployment

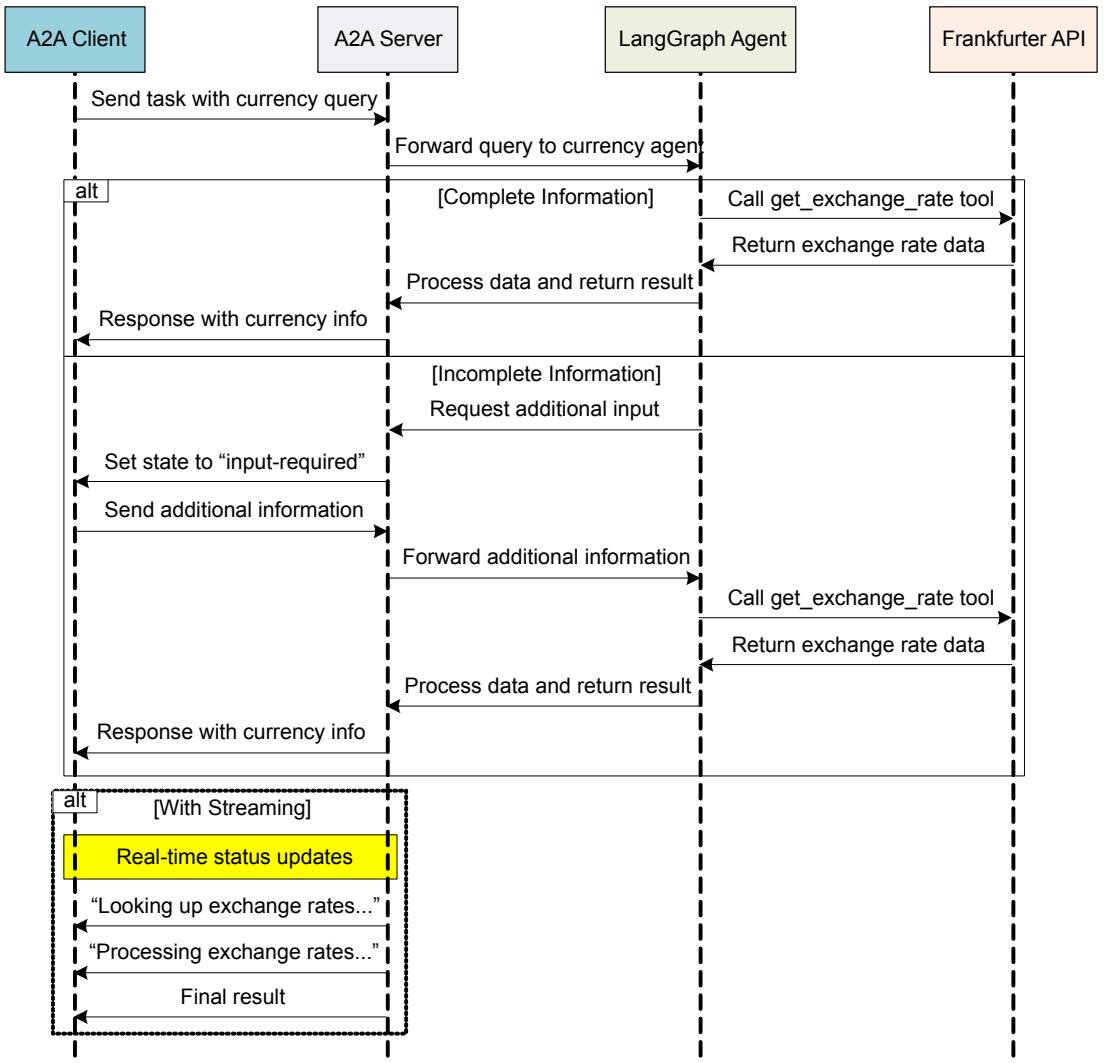

Fig. 5: LangGraph based A2A architecture deployment

and C3 AI[44] are leveraging A2A to accelerate everything from loan underwriting to fraud detection. At PayPal, a "risk-scoring" agent can feed transaction data to a "fraud-analysis" agent, which streams alerts back in sub-second bursts; then a "remediation" agent handles chargeback workflows automatically. Likewise, Intuit is prototyping agent chains that guide small-business owners through tax planning: a "data-aggregation" agent pulls bank statements, a "category-classifier" agent parses receipts, and a "compliance-agent" produces ready-to-file returns.

Manufacturing and supply-chain leaders such as SAP, Cotality[45] (formerly CoreLogic), and Quantiphi[46] are using A2A to orchestrate IoT-driven factory floors and dynamic logistics networks. SAP's "digital-twin" [75] agents coordinate predictive maintenance, quality inspection, and procurement: when a sensor-agent flags unusual vibration on a spindle, it triggers a "maintenance-planner" agent to schedule downtime, a "*Parts*-ordering" agent to requisition bearings, and a "production-planner" agent to reroute in-process batches. Cotality is exploring A2A in real-estate finance and insurance: underwriting, risk assessment, and portfolio management agents now share asset data, condition reports, and pricing models fluidly across back-office systems.

In customer-facing domains, ask-ai.com[47], Articul8[48], and GrowthLoop[49] are building multi-agent experiences that combine specialized bots for product recommendation, upsell coordination, and personalized support. Articul8's ModelMesh[50], acting as an "agent-of-agents," uses A2A to bring together domain-specific agents—e-commerce catalog, pricing engine, inventory tracker, and chatbot—to deliver a unified conversational storefront. GrowthLoop's marketing campaigns chain together content-generator, analytics-agent, and budget-optimizer agents to continuously refine targeting and creative in real time.

Security-focused AI infrastructure providers like Cohere[51] and Zeotap[52] are integrating A2A to ensure that inter-agent communications remain confidential and tamper-proof, even in air-gapped or highly regulated environments. Cohere's agents employ mutual TLS and JWT-signed *Task* payloads to authenticate peers before exchanging customer or medical data, while Zeotap is developing encrypted "data-masking" agents that share only anonymized telemetry with downstream partners.

Finally, niche innovators in fields as diverse as healthcare, legal, and education are beginning to pilot A2A proofs-of-concept. ServiceNow is experimenting with clinical-workflow agents that handle referral management, billing code validation, and patient follow-up notifications. Labelbox[53] and Databricks[54] are exploring training-data-curation agents that automatically label, validate, and version datasets before handing them off to model-training agents. Educational technology firms are assembling lesson-generator, assessment-agent, and tutor-bot networks to deliver adaptive, real-time feedback to learners at scale.

Together, this ecosystem illustrates how A2A is not merely another API standard but a foundational layer for the next generation of composable, highly automated enterprise applications. By decoupling capabilities into discoverable *Agent Cards*, orchestrating work as stateful *Tasks*, and exchanging richly typed *Parts* and *Artifacts*, organizations can build resilient, auditable, and infinitely extensible agent meshes tailored to every domain—from fintech and manufacturing to healthcare and customer experience.

In Figure 4, an A2A client issues a '*Tasks/send*' request containing a textual prompt to the A2A server [76]. The server transparently forwards that prompt to a specialized CrewAI[55] image-generation agent, which in turn calls the Google Gemini API to synthesize a picture. Once Gemini returns the finished image, the CrewAI agent stores it (e.g. in memory or on disk), hands back an *Artifact* reference to the A2A server, and the server finally responds to the original client with a *Task* result containing the image *Artifact*. This flow highlights how A2A's JSON-RPC interface and standardized *Task/Artifact* model let disparate components—clients, protocol gateways, domain-specific agents, and external services like Gemini—work together seamlessly to deliver on-demand image creation.

In another scenario (see Figure 5), an A2A client submits a currency-conversion request via a '*Tasks/send*' call to the A2A server, which forwards it to a LangGraph[56]-based agent [77]. If the initial query lacks details (for example, the target currency or date), the agent responds with an 'input-required' status and a follow-up prompt. The client then supplies the missing information in a subsequent '*Tasks/send*', and the agent invokes the Frankfurter API to retrieve live exchange rates. For streaming use cases, the client can instead issue a '*Tasks/sendSubscribe*' request—triggering an SSE stream of status updates ("Looking up rates…", "Processing…") before emitting the final '*TaskArtifact*' with the computed result. Throughout, the A2A protocol's *Task*, Message, and *Artifact* objects provide a consistent, stateful framework for multi-turn dialogues, real-time feedback, and webhook-style push notifications, making it easy to build conversational, long-running currency-conversion workflows.

Figure 6 illustrates the end-to-end architecture of an enterprise-grade A2A system, which is organized into three complementary tiers [78]. At the client tier, front-end SDKs or APIs interact with an A2A Client Agent responsible for encapsulating user requests and responses within the standardized A2A protocol. These protocol-compliant *Messages* are then routed to the runtime tier, embodied by the Agent Engine: a

[44] https://c3.ai/
[45] https://www.cotality.com/
[46] https://quantiphi.com/
[47] https://ai.com/
[48] https://www.articul8.ai/
[49] https://www.growthloop.com/
[50] https://www.articul8.ai/news/unleashing-the-next-frontier-of-enterprise-ai-introducing-model-mesh-dock-and-inter-lock-and-our-a2-a-partnership-with-google
[51] https://cohere.com/
[52] https://zeotap.com/
[53] https://labelbox.com/
[54] https://www.databricks.com/
[55] https://www.crewai.com/
[56] https://www.langchain.com/langgraph

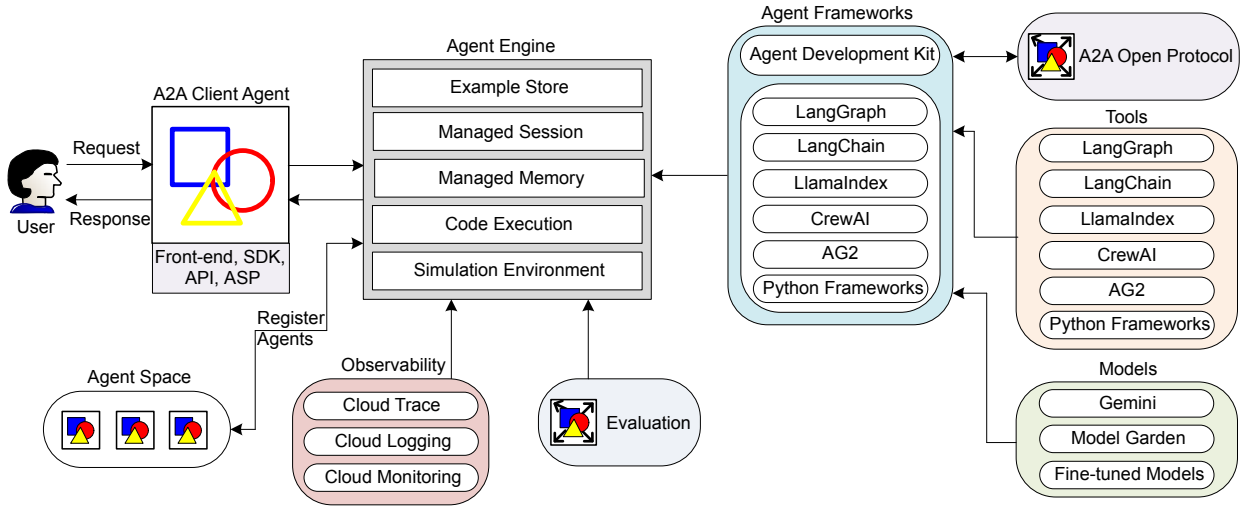

Fig. 6: GoogleADK based A2A architecture deployment

fully managed environment that performs agent registration, maintains an example repository, orchestrates session state and memory, executes sandboxed code, supports simulation workflows, and continuously emits telemetry into tracing, logging and monitoring pipelines while supplying performance data to evaluation modules. Finally, the development tier comprises the open Agent Development Kit (ADK)[57] together with frameworks such as LangGraph, LangChain[58], LlamaIndex[59], CrewAI and AG2—all speaking the same A2A "wire language." Agents built in this tier leverage large-language models (e.g. Gemini[60], Google's Model Garden offerings or custom fine-tuned checkpoints) to perform reasoning and tool invocation, enabling heterogeneous agents to discover, call and collaborate under a unified, enterprise-grade runtime.

Table X surveys a broad spectrum of early A2A adopters—from foundational enterprise platforms like Atlassian, Salesforce, and ServiceNow, through strategy and implementation partners such as Accenture, Deloitte, and BCG, to vertical specialists in finance (PayPal, Intuit), manufacturing (SAP, Cotality), and next-generation AI ecosystems (e.g. CrewAI, LangGraph, LlamaIndex, Semantic Kernel[61]). For each organization, we highlight the key domain where they're leveraging the A2A protocol—whether that's streamlining DevOps orchestration, automating IT service workflows, operationalizing machine-learning pipelines, enabling real-time fraud detection, or triggering self-healing observability chains—and pair it with a succinct "Problem – Solution – Vision" summary. This illustrates how standardized inter-agent task coordination and *Artifact* streaming address concrete challenges and unlock more autonomous, modular, and secure processes across diverse industry landscapes.

[57]https://github.com/google/A2A/blob/main/samples/python/agents/google_adk/README.md
[58]https://www.langchain.com/
[59]https://www.llamaindex.ai/
[60]https://gemini.google.com/?hl=en-IN
[61]https://github.com/microsoft/semantic-kernel

V. Key Challenges

The A2A protocol introduces a compelling abstraction for inter-agent collaboration. However, deploying A2A in real-world, production-grade systems reveals a diverse set of engineering and operational hurdles that go beyond protocol semantics. These challenges are not intrinsic flaws but reflect the realities of evolving schemas, multi-tenant authorization, secure message delivery, edge computing constraints, and regulatory enforcement in distributed systems. In this section, we focus on a curated subset of the most pressing challenges—specifically those that remain critical even as the ecosystem evolves—including schema evolution, secure push notifications, state persistence, high-throughput networking, data privacy, and distributed tracing. Our goal is to surface the infrastructural and architectural requirements necessary for scalable, resilient, and compliant A2A deployments in heterogeneous environments.

A. Schema Evolution

As the shared JSON-Schema underpinning A2A continually evolves, clients and agents that rigidly validate every incoming message risk breaking whenever a new optional parameter appears or an existing structure is augmented with extra properties [79]. To mitigate this, schema version identifiers must be explicitly declared in the *Agent Card*, signaling to consumers which version of the contract is in force. Both sides should implement tolerant deserialization that ignores unrecognized fields, logging them for operator review rather than failing the entire payload. In practice, this means layering a fallback parser atop the strict deserializer: unknown keys are stored in a generic map for potential use, while known fields drive the business logic. Continuous integration pipelines incorporate schema–code compatibility checks, automatically generating data classes with appropriate default values and guard clauses, and replaying recorded inter-agent interactions against the updated schema to catch regressions before they reach production [80]. The change management workflows can include automated diff reports for each schema update,

TABLE X: Comparison of Selected Industries Using Agent-to-Agent Protocol

Sector	**Organization**	**Domain of Work**	**Problem, Solution and Vision**
Enterprise Software	Atlassian	DevOps Orchestration	Agents in Jira/Confluence autonomously coordinate build, test, and release steps via A2A *Tasks*, eliminating custom point-to-point scripts.
Enterprise Software	Salesforce	CRM Automation	Sales, service, and marketing bots share leads, quotes, and case updates seamlessly through A2A, reducing manual handoffs.
Enterprise Software	ServiceNow	IT Service Management (ITSM) Workflows	Incident, change, and request-fulfillment agents collaborate across ticket queues and Configuration Management Database (CMDB) entries via A2A *Tasks* and *Artifacts*.
Content Management	Box	Document Collaboration	Storage and versioning agents exchange access-control decisions and metadata changes through A2A, enabling secure multi-user workflows
AI Platforms	C3 AI	Model Deployment	Cross-agent pipelines link predictive-model agents to data-preparation and monitoring agents using A2A for real-time operationalization
AI Platforms	Cohere	Secure Agent Meshes	Agents use mTLS and A2A *Tasks* to authenticate and encrypt inter-agent communications in air-gapped and regulated environments
Streaming	Confluent	Event-Driven Coordination	Kafka-connected agents publish and subscribe to Task events via A2A, enabling low-latency, event-driven business logic orchestration
Observability	Datadog	Self-Healing Systems	Diagnostic, allocator, and compliance agents form a remediation chain—triggering A2A *Tasks* on anomalies and streaming status updates over SSE
Observability	Elastic	Log-Driven Automation	Logging and alerting agents coordinate via A2A to index, analyze, and take corrective actions on log-based incidents
Consulting	Accenture	Digital Transformation	Multi-agent process blueprints decompose complex workflows into reusable A2A-orchestrated *Tasks* for end-to-end automation.
Consulting	Deloitte	Enterprise AI Strategy	Agents assess business risk, compliance, and performance metrics by exchanging domain-specific data and analysis via A2A.
Consulting	Cognizant	Industry-Specific Solutions	Agents tailored to banking, healthcare, and manufacturing collaborate across legacy systems using A2A Protocol to accelerate modernization.
Consulting	BCG	Intelligent Operations	Orchestration agents use A2A to tie together analytics engines, forecasting models, and execution bots for continuous process optimization.
Consulting	PwC	Risk & Compliance	Audit, controls, and reporting agents exchange evidence *Artifacts* via A2A to deliver unified, auditable compliance workflows.
FinTech	PayPal	Fraud Monitoring	Real-time risk-scoring and remediation agents share transaction insights via A2A, automatically blocking or flagging suspicious activity.
FinTech	Intuit	Tax & Accounting	Data-aggregation, classification, and compliance agents chain together via A2A to produce ready-to-file returns for SMBs.
Manufacturing	SAP	Shop-floor Automation	Digital-twin agents coordinate predictive maintenance, Parts ordering, and production-planning via A2A *Tasks* to minimize downtime.
Manufacturing	Cotality	Real-Estate Finance	Underwriting, valuation, and portfolio agents collaborate on property data using A2A to speed up loan approvals and risk assessments.
Marketing	GrowthLoop	Campaign Optimization	Content-generation, analytics, and budget-allocation agents iteratively refine digital campaigns through real-time A2A feedback loops.
Developer Tools	JetBrains	Coding Assistance	IDE-integrated agents provide code suggestions, refactoring, and security scans by delegating *Tasks* via A2A to specialized analysis bots.
Developer Tools	JFrog	Continuous Integration / Continuous Deployment (CI/CD) Reliability	Artifact-management and security-scan agents communicate over A2A to automate pipeline gating and vulnerability remediation.
Data Infrastructure	MongoDB	Hybrid Search	Search-index and recommendation agents coordinate via A2A to deliver low-latency, context-aware data retrieval across distributed clusters.
Data Infrastructure	Neo4j	Graph-Powered Retrieval-Augmented Generation (RAG)	GraphRAG agents query knowledge graphs and merge insights into A2A *Artifacts* for downstream analytic agents.
Observability	New Relic	Full-stack Insights	Monitoring, tracing, and anomaly-detection agents share event streams through A2A to trigger automated diagnostics and alerts.
Security	Zeotap	Data Privacy	Data-masking and consent-management agents utilize A2A to ensure only anonymized user data flows between analytics and marketing agents.
AI Framework Samples	CrewAI	Image Generation	A CrewAI agent generates images from text prompts via Gemini API; A2A handles request/response, artifact delivery, and caching.
AI Framework Samples	LangGraph	Currency Conversion	LangGraph ReAct agent integrates with Frankfurter API for live exchange rates, supports multi-turn dialogs, streaming, and push notifications.
AI Framework Samples	Genkit	Movie	Code Agents & TMDB-backed movie info agent and code-writing agent emit structured *Artifacts*, showcasing A2A for diverse content types.
AI Framework Samples	LlamaIndex	File Chat Workflow	File upload, parsing, and conversational QA agent with streaming updates, in-line citations, and webhook push notifications via A2A.
AI Framework Samples	Semantic Kernel	Travel	Planning & SK-based travel agent uses external plugins and APIs; A2A provides multi-turn, streaming, memory, and webhook notifications
AI Framework Samples	ADK	Expense Reimbursement	An ADK-based agent orchestrates a multi-turn reimbursement workflow: it prompts the client for missing expense details via dynamic webforms, ingests the completed form as a structured Artifact, and then finalizes the reimbursement task through A2A exchanges.

highlighting field additions, type changes, and deprecations, thereby enabling development teams to proactively adapt code and documentation in lockstep with specification changes. To further guard against drift, teams can adopt a contract testing framework where stubbed agents validate schema adherence prior to every merge.

B. Multi-Tenant Authentication

When a single A2A server instance spans multiple tenants or business units, it must enforce isolation at the HTTP layer by validating bearer tokens, mutual TLS certificates, or federation assertions. Upon each request, tokens are introspected via a central identity provider or verified against a known public key store, extracting claims for tenant identifiers, user roles, and permitted scopes. A dynamic policy engine—such as an embedded policy-as-code framework—evaluates each task invocation against rules that combine tenant context with declared agent skills. Policies are managed separately from code in a versioned repository, enabling live updates without redeploying agents [81], [82]. Detailed audit trails record which policy version and rule set applied to every call, providing forensic transparency and enabling compliance with internal and external regulations. The runtime policy caching can minimize latency by storing frequently used policy decisions in in-memory caches, balanced by short TTLs to reflect near-real-time policy changes. Integrating such a policy engine with A2A ensures that as organizational structures and compliance requirements evolve, the authorization logic adapts seamlessly.

C. Secure Push-Notification Configuration

Exposing a client-provided callback endpoint via the push notifications API without validation invites malicious actors to hijack notification traffic or flood arbitrary URLs [83]. A secure registration procedure employs a cryptographic handshake: the agent generates a nonce or validation token and performs an authenticated GET or HEAD request to the supplied URL with the token embedded as a query parameter or HTTP header. The client must echo the same token verbatim or sign it with a pre-shared key, proving ownership of the endpoint. Only after successful validation does the agent persist the webhook configuration. Furthermore, rate-limiting and exponential backoff prevent excessive notification bursts, while a circuit breaker automatically suspends misbehaving endpoints based on error thresholds. All registration and invocation events are recorded in immutable logs for later inspection. For more hardening, callback URLs can be bound to client certificates or ephemeral session tokens, ensuring that only authorized callbacks are honored and that replay attacks are thwarted. Such multi-layered verification protects agents from inadvertent participation in denial-of-service or spam campaigns.

D. State Persistence for Long-Running Tasks

Tasks that span extended durations cannot rely on in-memory data structures, as server restarts or container restarts would lose crucial context. Instead, each *Task*'s full state—status transitions, message history, *Artifact* updates, and pending sub*Tasks*—must be persisted to a durable store, such as a transactional database with append-only tables or an event streaming platform retaining ordered logs. When a client reconnects or resubscribes, the agent reconstructs the session by replaying stored events, preserving original timestamps and sequence ordering. Idempotent writes, enforced via composite keys on task identifier and event index, guarantee exactly-once delivery even under retry storms. Durable state backends also enable cross-agent recovery orchestration, allowing *Tasks* to resume seamlessly after failover. The sharded storage clusters can distribute task logs by tenant or workload to maintain high throughput and avoid single-node bottlenecks. Implementing such persistence layers ensures that even the most complex, multi-day workflows are robust against infrastructure disruptions.

E. High-Throughput, Low-Latency Networking

At scale, thousands of concurrent SSE streams and JSON-RPC calls can overwhelm naive HTTP servers and exhaust TLS handshakes. Deployments should leverage newer transport protocols—such as HTTP/2 or Quick UDP Internet Connections (QUIC) [84]—to multiplex multiple logical streams over shared connections, dramatically reducing handshake overhead. Clients and servers both implement connection pooling, keep-alive caching, and session resumption for TLS to minimize cryptographic costs. Back-pressure mechanisms throttle event emission when consumers fall behind, buffering a bounded queue of updates and signaling clients to reconcile state via synchronous retrieval calls. Also, strategically placed HTTP proxies or sidecar containers handle transport termination, freeing agent logic from low-level connection management and enabling horizontal scaling. Load testing tools simulate hundreds of thousands of concurrent sessions, tuning thread pools, event loop priorities, and buffer sizes to optimize for median and tail latencies under realistic workloads.

F. Distributed Tracing

In a mesh of interacting agents, diagnosing cross-agent performance issues or error cascades requires holistic traceability [85]. Propagating industry-standard trace context headers through HTTP calls and SSE streams enables end-to-end tracing of each *Task*'s lifecycle. Agents record spans annotated with key metadata—such as task identifier, agent role, invoked method, and outcome status—and export them to a centralized telemetry backend. Parallel structured logs capture enriched context (e.g. session identifiers, payload summaries, authentication claims), allowing teams to correlate traces with log events. Dashboards visualize latency percentiles, error rates, and dependency topologies, alerting on anomalies and enabling rapid root-cause isolation. Integration with service meshes can provide automatic trace injection at the network layer, further reducing instrumentation burden on agent code. Comprehensive observability transforms A2A from a black-box protocol into a transparent, diagnosable platform [86].

G. Data Compliance

Tasks often carry sensitive or regulated data—ranging from personal identifiers to financial transactions. To safeguard confidentiality, agents and clients apply message-level encryption using JSON Web Encryption, with keys provisioned securely from a centralized key vault. Large *Artifacts* (i.e. binary files or multimedia) may be stored in encrypted object stores, with A2A payloads containing only secure reference URIs. Immutable audit logs, supporting write-once, read-many semantics, chronicle every state transition and data access for compliance audits under frameworks like GDPR or HIPAA. Dynamic redaction policies enforce field-level masking based on tenant or user roles, filtering sensitive content before any logging or forwarding steps. Periodic compliance scanning workflows validate that no sensitive fields are leaked in logs or metrics, and data retention policies automatically purge expired records in accordance with legal requirements [87], [88], [89].

H. Semantic Interoperability

Different agents may describe identical concepts using divergent field names or schemas—for instance, using *userId* versus *actor_id*. To achieve true semantic interoperability, enterprises adopt a shared ontology expressed as JSON-LD contexts or Shapes Constraint Language (SHACL) shapes [90]. Agents load these definitions at startup and apply transformation layers that map external field names into canonical internal representations. Code generation tools produce adapters that enforce these mappings at compile time, ensuring consistent interpretation across the agent ecosystem. Regular conformance tests validate that each agent's published schema and runtime behavior align with the shared ontology, preventing silent data mismatches in complex cross-agent workflows. Versioned ontology registries allow agents to negotiate and agree on which ontology version to use, ensuring backward compatibility and graceful migrations [91].

I. Edge-Deployed Agents

Edge and IoT deployments often lack the CPU and memory capacity for full JSON parsing and TLS termination. To accommodate these constraints, edge agents implement a lean subset of A2A using compact binary encodings (e.g. Concise Binary Object Representation (CBOR)[92] or MessagePack[93]) and delegate heavy transports to a proxied gateway service. Core RPC methods—such as *Tasks/send* and *Tasks/get*—remain available locally, while streaming and push notifications are handled by a cloud gateway. WebAssembly[62]-based runtimes host these lightweight agents, providing secure sandboxing and predictable resource usage on constrained hardware, without sacrificing interoperability with full-featured A2A implementations. Additionally, renewable credentials and dynamic configuration updates can be pushed to edge nodes via lightweight heartbeat protocols, ensuring that security and discovery remain synchronized across disconnected environments [94], [95].

[62]https://webassembly.org/

J. Regulatory Auditability

In heavily regulated sectors, every inter-agent interaction must be provably auditable, traceable, and tamper-evident [96]. Agents employ Federal Information Protection Standard (FIPS)-validated cryptographic modules for key operations, store all *Task* life cycle events and payload digests in append-only, write-once storage, and expose dedicated conformance endpoints for external auditors. A formal certification suite—maintained by the A2A community—verifies compliance with each required JSON-RPC method, SSE event type, error code, and security control. Certified implementations bear an "A2A Conformant" seal, providing enterprises with assurance that cross-vendor workflows meet stringent audit and regulatory demands. Periodic re-certification exercises and live conformance tests guard against drift, ensuring that security guarantees endure through continuous protocol evolution [97].

VI. Future Directions

As A2A matures and sees broader adoption across decentralized and hybrid infrastructures, there is a clear need to push beyond the foundational capabilities. The next phase of A2A evolution will be shaped by enhancements that address current bottlenecks, unlock edge and AI-native deployments, and fortify the protocol's security and governance frameworks. In this section, we present a focused set of high-impact future directions—ranging from automated schema migration and decentralized discovery to serverless execution, meta-agent intelligence, and confidential computing. These proposals are not speculative blueprints but practical, extensible improvements rooted in unmet needs surfaced through early deployments. Together, they chart a path for turning A2A into a universally adaptable, context-aware, and privacy-preserving agent communication standard.

A. Automated Schema Migration Tooling

Future toolchains will deliver end-to-end automation for evolving A2A schemas by analyzing differences between *Agent Card* versions and producing migration adapters. A command-line utility will generate safe default values for newly introduced fields, flag deprecated properties for review, and scaffold transformation functions in client and server code. This system integrates with type-safe languages to enforce compile-time checks and provides runtime fallbacks for unknown properties. Continuous integration pipelines can then execute these migration steps automatically, ensuring that schema changes do not break deployed agents and that compatibility across versions is maintained without manual intervention [98]. Additionally, interactive conflict resolution prompts will guide developers through ambiguous migrations, preserving data integrity.

B. Decentralized P2P Agent Discovery

To remove single points of failure and enable truly federated networks, A2A can incorporate decentralized discovery mechanisms such as Distributed Hash Tables or permissioned

blockchain anchors. Agents publish cryptographic hashes of their *Agent Cards* to the Distributed Hash Table (DHT) [99] under namespace keys derived from their domain names. Peers query the DHT to retrieve hashes, then fetch and verify signed *Agent Cards* from their endpoints. On-chain smart contracts record version history and access control policies, guaranteeing immutability and auditability. This model allows agents across administrative boundaries to discover each other securely, dynamically, and without relying on centralized registries or DNS, greatly enhancing resilience and flexibility in multi-organizational ecosystems. Gossip-based fallbacks ensure rapid propagation in local clusters.

C. Attribute-Based Access Control Integration

Embedding detailed ABAC policies directly within the *Agent Card* allows agents to enforce fine-grained permissions locally [100]. Policies authored in a language like "Rego" [101] can reference user claims—such as department, role, or project membership—extracted from bearer tokens or mTLS certificates. Upon receiving a *Task*, the agent invokes its in-process policy engine to evaluate whether the request meets the specified conditions, returning a deny or allow decision before executing any business logic. Policy bundles are versioned and fetched from a secure registry at startup, and hot-reloading capabilities enable immediate enforcement of updated policies. This approach decentralizes authorization, simplifies audit trails, and reduces external dependencies on policy servers. Metrics on policy decision frequency help detect stale rules.

D. Unified Notification Broker Service

Rather than implementing separate streams for SSE, webhooks, and message queues, a cloud-native A2A Push Notification Broker can unify all transports under a single publish-subscribe interface [102], [103]. Agents emit *Task* updates to the broker, which then fans out *Messages* to subscribers via the transport of their choice—SSE endpoints, email, collaboration tools, or enterprise messaging platforms. Built-in retry logic, dead-letter queues, and schema validation ensure reliable delivery and observability. Clients register once with the broker and receive updates regardless of the underlying transport, decoupling agents from notification complexities and centralizing control over routing policies, rate limits, and monitoring dashboards. Per-tenant isolation mechanisms and audit logs bolster security compliance.

E. Serverless Edge-Native Agents

Packaging agents as serverless functions or WebAssembly modules enables elastic scaling [103], [104], [105] and secure sandboxing at the edge [106]. Short-lived *Tasks* trigger on-demand scaling, while long-running workflows checkpoint state to durable storage between invocations. Edge agents, running in lightweight runtimes on gateways or appliances, process low-latency requests locally and buffer outgoing *Task* updates during network interruptions. A hybrid orchestration model synchronizes state with cloud backends when connectivity resumes. This serverless and edge-native paradigm reduces operational overhead, optimizes resource utilization, and supports distributed scenarios—such as IoT telemetry processing and mobile offline interactions—without sacrificing the full power of the A2A protocol. Cold-start reductions improve responsiveness on resource-constrained devices.

F. AI-Driven Optimization of Meta-Agent

A meta-agent leveraging machine learning can optimize task routing and resource allocation across the A2A mesh. By continuously ingesting telemetry—latency, error rates, resource utilization—it trains policies that predict which agent instance will respond most efficiently to a new *Task*. The meta-agent proactively pre-warms idle instances, scales clusters in anticipation of workload spikes, and rebalances traffic to minimize tail latency. Feedback loops update routing strategies based on real-time performance data, allowing the mesh to self-adapt under changing conditions. This AI-driven orchestration improves throughput, reduces time to completion, and ensures high availability without manual tuning. Anomaly detection triggers automated remediation workflows.

G. Multi-Modal Streaming Protocols

To support rich media *Artifacts*, future A2A streams will adopt WebRTC DataChannels and QUIC for low-latency, bidirectional transfers of audio, video, and binary data. Agents negotiate media parameters—such as codecs, bitrates, and packetization schemes—in an extended capability handshake prior to streaming. HTTP/3's datagram APIs can multiplex multiple *Artifact* streams concurrently, preventing head-of-line blocking when one large transfer stalls. Adaptive bitrate algorithms dynamically adjust media quality based on network conditions, and fallback mechanisms ensure that agents can gracefully degrade to text or compressed representations when high-bandwidth transports are unavailable. Forward error correction enhances stream reliability over lossy links.

H. Confidential Computing

To protect sensitive data on untrusted infrastructure, agents will run inside Trusted Execution Environments (TEEs) provided by technologies such as Intel Software Guard Extensions (SGX)[63] or AMD Secure Encrypted Virtualization(SEV)[64]. Task inputs and *Artifacts* remain encrypted at rest and in transit, only decrypted within the enclave boundary. Remote attestation protocols allow clients to verify the integrity and authenticity of the enclave before sending data. Key management integrates with hardware-backed key vaults, ensuring that cryptographic materials never leave the secure enclave. This architecture satisfies stringent confidentiality and compliance requirements—such as those found in finance and healthcare—without compromising interoperability. Side-channel mitigations ensure robust enclave security.

[63]https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html

[64]https://www.amd.com/en/developer/sev.html

I. AI-Native Load Balancing

Moving beyond static routing algorithms, AI-native load balancers will employ predictive models trained on historical telemetry—such as resource utilization patterns, error distributions, and cold-start latencies—to route incoming *Tasks* to the agent instance most likely to meet performance targets. Feature inputs include current CPU and memory usage, recent throughput trends, and network latency measurements [107], [108]. As conditions evolve, the AI-driven balancer continuously retrains and updates routing policies, ensuring that task placement adapts in real time to shifting workloads and infrastructure changes, thereby minimizing tail latency and maximizing resource efficiency. Explainable AI techniques provide transparency into routing decisions.

J. Standardized Certification Suite

A formal conformance test harness, maintained under the A2A working group, will include reference implementations that simulate edge, cloud, and high-throughput scenarios [109], [110]. Vendors download Dockerized test kits to validate compliance with JSON-RPC method specifications, SSE event sequences, error-code mappings, and performance benchmarks. The suite also verifies integration with transport security features, such as mTLS and OAuth flows. Passing the conformance tests grants an official "A2A Certified" badge, instilling confidence in multi-vendor interoperability and accelerating enterprise adoption by providing a trusted interoperability guarantee [111].

VII. Conclusion

This article has explored the foundations, applications, and challenges of orchestrating autonomous agents through the A2A protocol. By introducing a unified *Task*-and-*Artifact* abstraction layered over HTTP, JSON-RPC, and event-streaming mechanisms, A2A offers a compelling alternative to brittle, ad-hoc integrations that have historically constrained multi-agent interoperability. Through a detailed examination of real-world use cases spanning enterprise automation, smart manufacturing, healthcare coordination, and disaster response, this review has illuminated both the opportunities and complexities inherent in deploying A2A at scale. In addressing critical deployment challenges—ranging from schema evolution and dynamic policy enforcement to durable state persistence and semantic alignment—we have outlined a robust set of best practices and architectural strategies.

References

[1] Hexmoor H, Lammens J, Caicedo G, Shapiro SC. Behaviour based AI, cognitive processes, and emergent behaviors in autonomous agents. WIT Press; 2025 Jan 22.

[2] Liu Z, Zhao D, Yuan B, Jiang Z. Rescueadi: Adaptive disaster interpretation in remote sensing images with autonomous agents. IEEE Transactions on Geoscience and Remote Sensing. 2025 Jan 22.

[3] Ramos MC, Collison CJ, White AD. A review of large language models and autonomous agents in chemistry. Chemical Science. 2025.

[4] Papaioannou S, Kolios P, Theocharides T, Panayiotou CG, Polycarpou MM. Rolling horizon coverage control with collaborative autonomous agents. Philosophical Transactions A. 2025 Jan 30;383(2289):20240146.

[5] Acharya DB, Kuppan K, Divya B. Agentic AI: Autonomous Intelligence for Complex Goals–A Comprehensive Survey. IEEE Access. 2025 Jan 22.

[6] Chang WC, Esmaeili B, Hasanzadeh S. Impacts of Physical and Informational Failures on Worker–Autonomy Trust in Future Construction. Journal of Construction Engineering and Management. 2025 Apr 1;151(4):04025011.

[7] Zhang Y, Ma Z, Ma Y, Han Z, Wu Y, Tresp V. Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. InProceedings of the AAAI Conference on Artificial Intelligence 2025 Apr 11 (Vol. 39, No. 22, pp. 23378-23386).

[8] Ho CT, Ren H, Khailany B. Verilogcoder: Autonomous verilog coding agents with graph-based planning and abstract syntax tree (ast)-based waveform tracing tool. InProceedings of the AAAI Conference on Artificial Intelligence 2025 Apr 11 (Vol. 39, No. 1, pp. 300-307).

[9] Addison U. La VIDA: towards a motivated goal reasoning agent. Autonomous Agents and Multi-Agent Systems. 2025 Jun;39(1):5.

[10] Ning H, Li Z, Akinboyewa T, Lessani MN. An autonomous GIS agent framework for geospatial data retrieval. International Journal of Digital Earth. 2025 Dec 31;18(1):2458688.

[11] Akl J, Gning A, Omrani H, Caspary O, Blansché A, Abdallah F. Opportunistic collaboration between heterogeneous agents using an unstructured ontology via GenAI. International Journal of Intelligent Robotics and Applications. 2025 Jan 17:1-3.

[12] Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, Chen Z, Tang J, Chen X, Lin Y, Zhao WX. A survey on large language model based autonomous agents. Frontiers of Computer Science. 2024 Dec;18(6):186345.

[13] Platas-Lopez A, Guerra-Hernandez A, Quiroz-Castellanos M, Cruz-Ramirez N. A survey on agent-based modelling assisted by machine learning. Expert Systems. 2025 Jan;42(1):e13325.

[14] Anjomshoae S, Najjar A, Calvaresi D, Främling K. Explainable agents and robots: Results from a systematic literature review. In18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019 2019 (pp. 1078-1088). International Foundation for Autonomous Agents and Multiagent Systems.

[15] Barua S. Exploring autonomous agents through the lens of large language models: A review. arXiv preprint arXiv:2404.04442. 2024 Apr 5.

[16] Kampik T, Mansour A, Boissier O, Kirrane S, Padget J, Payne TR, Singh MP, Tamma V, Zimmermann A. Governance of autonomous agents on the web: Challenges and opportunities. ACM Transactions on Internet Technology. 2022 Nov 14;22(4):1-31.

[17] Sado F, Loo CK, Liew WS, Kerzel M, Wermter S. Explainable goal-driven agents and robots-a comprehensive review. ACM Computing Surveys. 2023 Feb 2;55(10):1-41.

[18] Sakai T, Nagai T. Explainable autonomous robots: a survey and perspective. Advanced Robotics. 2022 Mar 19;36(5-6):219-38.

[19] Kiran BR, Sobh I, Talpaert V, Mannion P, Al Sallab AA, Yogamani S, Pérez P. Deep reinforcement learning for autonomous driving: A survey. IEEE transactions on intelligent transportation systems. 2021 Feb 9;23(6):4909-26.

[20] Agent2Agent Protocol (A2A), https://google.github.io/A2A/

[21] Ingle A., Using Google's Agent Development Kit and Agent2Agent, https://wandb.ai/gladiator/Google-Agent2Agent/reports/Tutorial-Using-Google-s-Agent2Agent-A2A-protocol–VmlldzoxMjIyODEwOA

[22] Davies A., Davies https://wandb.ai/onlineinference/mcp/reports/Google-s-Agent2Agent-A2A-protocol-A-new-standard-for-AI-agent-collaboration–VmlldzoxMjIxMTk1OQ

[23] Mikami L., What is The Agent2Agent Protocol (A2A) and Why You Must Learn It Now, https://huggingface.co/blog/lynn-mikami/agent2agent

[24] Sharma A., Agent-to-Agent Protocol: Helping AI Agents Work Together Across Systems, https://community.analyticsvidhya.com/c/generative-ai-tech-discussion/agent-to-agent-protocol-helping-ai-agents-work-together-across-systems

[25] Langley, B.K., Paolucci, M. and Sycara, K., 2003, July. Discovery of infrastructure in multi-agent systems. In Proceedings of the second international joint conference on Autonomous agents and multiagent systems (pp. 1046-1047).

[26] Sohn, S.S., Lee, M., Moon, S., Qiao, G., Usman, M., Yoon, S., Pavlovic, V. and Kapadia, M., 2022. A2X: An end-to-end framework for assessing agent and environment interactions in multimodal human trajectory prediction. Computers & Graphics, 106, pp.130-140.

[27] Sohn, S.S., Lee, M., Moon, S., Qiao, G., Usman, M., Yoon, S., Pavlovic, V. and Kapadia, M., 2021, November. A2x: An agent and environment interaction benchmark for multimodal human trajectory prediction. In

Proceedings of the 14th ACM SIGGRAPH Conference on Motion, Interaction and Games (pp. 1-9).
[28] Bhattacharyya, P., Huang, C. and Czarnecki, K., 2024, June. Ssl-interactions: Pretext *Tasks* for interactive trajectory prediction. In 2024 IEEE Intelligent Vehicles Symposium (IV) (pp. 1450-1457). IEEE.
[29] Tedeschini, B.C., Brambilla, M., Nicoli, M. and Win, M.Z., 2024, July. Cooperative Positioning with Multi-Agent Reinforcement Learning. In 2024 27th International Conference on Information Fusion (FUSION) (pp. 1-7). IEEE.
[30] Tedeschini, B.C., Brambilla, M., Nicoli, M. and Win, M.Z., 2024. Multi-agent Reinforcement Learning for Distributed Cooperative Vehicular Positioning. IEEE Transactions on Intelligent Vehicles.
[31] Bosse, S. and Lehmhus, D., 2019. Material-integrated cluster computing in self-adaptive robotic materials using mobile multi-agent systems. Cluster Computing, 22(3), pp.1017-1037.
[32] Minelli, G., Decentralised Coordination and Communication in Multi-Agent Reinforcement Learning Systems. Thesis. URL: https://amslaurea.unibo.it/id/eprint/28531
[33] Surapaneni, R., Jha, M., Vakoc, M., Segal, T., Announcing the Agent2Agent Protocol (A2A), URL: https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/
[34] Ghosh, D.P., Agentic Ecosystemin Engineering Design: A Framework for Interoperable Legacy Tools and Emergent Collaboration via MCP/A2A Protocols. 2025. ResearchGate. Preprint. Web URL: https://www.researchgate.net/publication/390667861. DOI: 10.13140/RG.2.2.27720.64008
[35] Ghosh, A., Google's A2A Protocol: Here's What You Need to Know, URl: https://learnopencv.com/googles-a2a-protocol-heres-what-you-need-to-know/
[36] Mattson, E., Integrating Semantic Kernel Python with Google's A2A Protocol, URL: https://devblogs.microsoft.com/foundry/semantic-kernel-a2a-integration/
[37] Jellema, L., A2A (Agent-to-Agent) is not about AI, URL: https://lucasjellema.medium.com/a2a-agent-to-agent-is-not-about-ai-f814c0ac0682
[38] Chawla A., A Visual Guide to Agent2Agent (A2A) Protocol (it does not compete with MCPs), URL: https://blog.dailydoseofds.com/p/a-visual-guide-to-agent2agent-a2a
[39] Lee, V., and Lei, A., What Is Google's Agent2Agent (A2A) Protocol? A Complete Guide to AI Interoperability, UEL: https://www.byteplus.com/en/blog/what-is-google-agent-2-agent-protocol
[40] Mukhopadhyay, B., Google's Agent2Agent Protocol and Python A2A: Revolutionizing AI Interoperability, https://medium.com/devdotcom/googles-agent2agent-protocol-and-python-a2a-revolutionizing-ai-interoperability-5dca7ed091e8
[41] Ichhpurani, K., Building the industry's best agentic AI ecosystem with partners, URL: https://cloud.google.com/blog/topics/partners/best-agentic-ecosystem-helping-partners-build-ai-agents-next25
[42] Google's Agent2Agent interoperability protocol aims to standardize agentic communication, UEL: https://venturebeat.com/ai/googles-agent2agent-interoperability-protocol-aims-to-standardize-agentic-communication/
[43] Lardinois, F., Google's Agent2Agent Protocol Helps AI Agents Talk to Each Other, https://thenewstack.io/googles-agent2agent-protocol-helps-ai-agents-talk-to-each-other/
[44] A2A in Action, https://storage.googleapis.com/gweb-developer-goog-blog-assets/original_videos/A2A_demo_v4.mp4
[45] A2A vs MCP, https://google.github.io/A2A/#/topics/a2a_and_mcp
[46] Benji N., A2A Vs. MCP - Which Agentic Framework To Pick?, https://blog.gopenai.com/a2a-vs-mcp-which-agentic-framework-to-pick-73dad0f21c24
[47] Arsanjani A., Complementary Protocols for Agentic Systems : Understanding Google's A2A & Anthropic's MCP, https://dr-arsanjani.medium.com/complementary-protocols-for-agentic-systems-understanding-googles-a2a-anthropic-s-mcp-47f5e66b6486
[48] Ollama, https://ollama.com/
[49] Llama.cpp, https://github.com/ggml-org/llama.cpp
[50] GPT4All, https://www.nomic.ai/gpt4all
[51] vLLM, https://docs.vllm.ai/
[52] Mishra, H., Agent-to-Agent Protocol: Helping AI Agents Work Together Across Systems, https://www.analyticsvidhya.com/blog/2025/04/agent-to-agent-protocol/
[53] Gupta, H., Understanding Google's A2A Protocol: A New Era of Agent-to-Agent Communication, https://medium.com/@devharshgupta.com/understanding-googles-a2a-protocol-a-new-era-of-agent-to-agent-communication-e7ae92ba0ece
[54] A2A Protocol Introduction, https://www.a2aprotocol.net/docs/introduction
[55] Ji, X., Google A2A protocol for healthcare systems: revolutionizing AI agent communication, https://www.byteplus.com/en/topic/551097?title=google-a2a-protocol-for-healthcare-systems-revolutionizing-ai-agent-communication
[56] Desai, G., Comparing Model Context Protocol (MCP) and Agent-to-Agent (A2A) Protocol: Complementary Patterns in AI, https://www.linkedin.com/pulse/comparing-model-context-protocol-mcp-agent-to-agent-a2a-gaurang-desai-rxu1e/
[57] Tanghe, L., A2A - Breaking the barriers of e-commerce, http://linkedin.com/pulse/a2a-breaking-barriers-e-commerce-lucien-tanghe-wguoe/
[58] Nikpour M, Yousefi PB, Jafarzadeh H, Danesh K, Shomali R, Asadi S, Lonbar AG, Ahmadi M. Intelligent energy management with iot framework in smart cities using intelligent analysis: An application of machine learning methods for complex networks and systems. Journal of Network and Computer Applications. 2025 Mar 1;235:104089.
[59] Siva Rama Krishna U, Vasudeva Pavan Kumar N, Tadi C, H. Badiger M. Internet of things and digital twins for future smart cities: scientometric analysis. Intelligent Buildings International. 2025 Feb 11:1-3.
[60] Tripathy S, Swaminathan K, Bhadoriya S, Sapre AA, Singh S. Evolving Dynamics: Online Contracts in the Era of Artificial Intelligence. InCutting-Edge Technologies for Business Sectors 2025 (pp. 245-270). IGI Global.
[61] Keykhaei M, Samany NN, Jelokhani-Niaraki M, Zlatanova S. Multi-agent-based human cognition simulation of Situation-aware earthquake emergency evacuation. International Journal of Disaster Risk Reduction. 2024 Jan 1;100:104183.
[62] Sako DJ, Igiri CG, Bennett EO, Deedam FB. ESARS: A situation-aware multi-agent system for real-time emergency response management. European Journal of Information Technologies and Computer Science. 2024 Feb 9;4(1):1-8.
[63] Zekhnini K, Chaouni Benabdellah A, Cherrafi A. A multi-agent based big data analytics system for viable supplier selection. Journal of Intelligent Manufacturing. 2024 Dec;35(8):3753-73.
[64] Bakopoulos E, Siatras V, Mavrothalassitis P, Nikolakis N, Alexopoulos K. Digital-twin-enabled framework for training and deploying AI agents for production scheduling. InArtificial Intelligence in Manufacturing: Enabling Intelligent, Flexible and Cost-Effective Production Through AI 2024 Feb 9 (pp. 147-179). Cham: Springer Nature Switzerland.
[65] Heinrich C. Transport layer security (TLS). InEncyclopedia of Cryptography, Security and Privacy 2025 Jan 8 (pp. 2645-2646). Cham: Springer Nature Switzerland.
[66] Disable-TlsCipherSuite (TLS), https://learn.microsoft.com/en-us/powershell/module/tls/disable-tlsciphersuite?view=windowsserver2025-ps
[67] enforce forward secrecy suites, https://www.digicert.com/kb/ssl-support/ssl-enabling-perfect-forward-secrecy.htm
[68] Morales DA, Wazan AS, Chadwick DW, Laborde R, Maramara AR. Enhancing the ACME protocol to automate the management of all X. 509 web certificates (Extended version). Computer Communications. 2025 Apr 15;236:108106.
[69] X.509 Internet Public Key Infrastructure: Online Certificate Status Protocol - OCSP, https://www.rfc-editor.org/rfc/rfc6960
[70] Certificate revocation list (CRL), https://en.wikipedia.org/wiki/Certificate_revocation_list
[71] Use Cases and Requirements for DNS-Based Authentication of Named Entities (DANE), https://www.rfc-editor.org/rfc/rfc6394
[72] Indicating Resolver Support of DNSSEC, https://www.rfc-editor.org/rfc/rfc3225
[73] Cloud log analytics, https://cloud.google.com/logging/docs/log-analytics
[74] Levai T, Retvari G. Batch-scheduling Data Flow Graphs with Service-level Objectives on Multicore Systems. Infocommunications Journal. 2022;14(1):43-50.
[75] Adenekan TK. The Role of Digital Twins in Enhancing SAP-Based Supply Chain Optimization. URL: https://www.researchgate.net/publication/388629044
[76] CrewAI Agent with A2A Protocol, https://github.com/google/A2A/blob/main/samples/python/ agents/crewai/README.md
[77] LangGraph Currency Agent with A2A Protocol, https://github.com/google/A2A/blob/main/samples/python/agents/langgraph/README.md
[78] Vertex AI offers new ways to build and manage multi-agent systems, https://cloud.google.com/blog/products/ai-machine-learning/build-and-manage-multi-system-agents-with-vertex-ai

[79] Levi E, Kadar I. IntellAgent: A Multi-Agent Framework for Evaluating Conversational AI Systems. arXiv preprint arXiv:2501.11067. 2025 Jan 19.

[80] Rissaki A, Fountalis I, Vasiloglou N, Gatterbauer W. Towards Agentic Schema Refinement. arXiv preprint arXiv:2412.07786. 2024 Nov 25.

[81] Zhu S, Lu J, Lyu B, Pan T, Zhang S, Sun X, Jia C, Cheng X, Kang D, Lv Y, Yang F. Proactive Telemetry in Large-Scale Multi-Tenant Cloud Overlay Networks. IEEE/ACM Transactions on Networking. 2024 Apr 4.

[82] Gama Garcia A, Alcaraz Calero JM, Mora Mora H, Wang Q. ServiceNet: resource-efficient architecture for topology discovery in large-scale multi-tenant clouds. Cluster Computing. 2024 Oct;27(7):8965-82.

[83] Sombattheera C. News Feed: A Multiagent-Based Push Notification System. InInternational Conference on Multi-disciplinary Trends in Artificial Intelligence 2022 Nov 10 (pp. 120-132). Cham: Springer International Publishing.

[84] Simpson A, Alshaali M, Tu W, Asghar MR. Quick UDP Internet Connections and Transmission Control Protocol in unsafe networks: A comparative analysis. IET Smart Cities. 2024 Dec;6(4):351-60.

[85] Chan A, Ezell C, Kaufmann M, Wei K, Hammond L, Bradley H, Bluemke E, Rajkumar N, Krueger D, Kolt N, Heim L. Visibility into AI agents. InProceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency 2024 Jun 3 (pp. 958-973).

[86] Li B, Peng X, Xiang Q, Wang H, Xie T, Sun J, Liu X. Enjoy your observability: an industrial survey of microservice tracing and analysis. Empirical Software Engineering. 2022 Jan;27:1-28.

[87] Alves PH, Correia F, Frajhof I, De Souza CS, Lopes H. Designing intelligent agents in normative systems toward data regulation representation. IEEE Access. 2023 May 15;11:51590-605.

[88] Rayarao SR. AI-Powered Agents Transforming 401 (k) Compliance and Risk Management. Authorea Preprints. 2025 Jan 5.

[89] Roger J, Alexander D. AI-Powered Risk Assessment Models for Enhancing Data Governance Compliance. URL: https://www.researchgate.net/publication/390941575

[90] SHACL, https://www.w3.org/TR/2016/WD-shacl-20160128/

[91] Nilsson J, Javed S, Albertsson K, Delsing J, Liwicki M, Sandin F. Ai concepts for system of systems dynamic interoperability. Sensors. 2024 May 3;24(9):2921.

[92] Concise Binary Object Representation, https://datatracker.ietf.org/doc/html/rfc8949

[93] MessagePack, https://msgpack.org/index.html

[94] Nandhakumar AR, Baranwal A, Choudhary P, Golec M, Gill SS. EdgeAISim: A toolkit for simulation and modelling of AI models in edge computing environments. Measurement: Sensors. 2024 Feb 1;31:100939.

[95] Gill SS, Golec M, Hu J, Xu M, Du J, Wu H, Walia GK, Murugesan SS, Ali B, Kumar M, Ye K. Edge AI: A taxonomy, systematic review and future directions. Cluster Computing. 2025 Feb;28(1):1-53.

[96] Cohen MK, Kolt N, Bengio Y, Hadfield GK, Russell S. Regulating advanced artificial agents. Science. 2024 Apr 5;384(6691):36-8.

[97] Afroogh S, Akbari A, Malone E, Kargar M, Alambeigi H. Trust in AI: progress, challenges, and future directions. Humanities and Social Sciences Communications. 2024 Nov 18;11(1):1-30.

[98] Wang X, Li B, Song Y, Xu FF, Tang X, Zhuge M, Pan J, Song Y, Li B, Singh J, Tran HH. Openhands: An open platform for ai software developers as generalist agents. InThe Thirteenth International Conference on Learning Representations 2024 Oct.

[99] Gaba S, Khan H, Almalki KJ, Jabbari A, Budhiraja I, Kumar V, Singh A, Singh KK, Askar SS, Abouhawwash M. Holochain: An agent-centric distributed hash table security in smart IoT applications. IEEE Access. 2023 Jul 31;11:81205-23.

[100] Zaidi SY, Shah MA, Khattak HA, Maple C, Rauf HT, El-Sherbeeny AM, El-Meligy MA. An attribute-based access control for IoT using blockchain and smart contracts. Sustainability. 2021 Sep 23;13(19):10556.

[101] Rego, https://www.openpolicyagent.org/docs/latest/policy-language/

[102] Di Benedetto G. " Agent Code, James Code": AI-based code generation framework (Doctoral dissertation, Politecnico di Torino).https://webthesis.biblio.polito.it/secure/31756/1/tesi.pdf

[103] Saleh A, Morabito R, Tarkoma S, Pirttikangas S, Lovén L. Towards message brokers for generative ai: Survey, challenges, and opportunities. arXiv preprint arXiv:2312.14647. 2023 Dec 22.

[104] Samdani G, Paul K, Saldanha F. Serverless architectures for agentic AI deployment. https://doi.org/10.30574/wjaets.2022.7.2.0144

[105] Singh AK, Kumar S, Jain S. A multi-agent deep reinforcement learning approach for optimal resource management in serverless computing. Cluster Computing. 2025 Apr;28(2):102.

[106] Lin J, Zhao H, Zhang A, Wu Y, Ping H, Chen Q. Agentsims: An open-source sandbox for large language model evaluation. arXiv preprint arXiv:2308.04026. 2023 Aug 8.

[107] Hong S, Zheng X, Chen J, Cheng Y, Wang J, Zhang C, Wang Z, Yau SK, Lin Z, Zhou L, Ran C. Metagpt: Meta programming for multi-agent collaborative framework. arXiv preprint arXiv:2308.00352. 2023 Nov;3(4):6.

[108] Wang JX. Meta-learning in natural and artificial intelligence. Current Opinion in Behavioral Sciences. 2021 Apr 1;38:90-5.

[109] Blosser M, Weihrauch A. A consumer perspective of AI certification–the current certification landscape, consumer approval and directions for future research. European Journal of Marketing. 2024 Feb 8;58(2):441-70.

[110] Cihon P, Kleinaltenkamp MJ, Schuett J, Baum SD. AI certification: Advancing ethical practice by reducing information asymmetries. IEEE Transactions on Technology and Society. 2021 May 10;2(4):200-9.

[111] Rampasek M, Mesarck M, Andrasko J. Evolving cybersecurity of AI-featured digital products and services: Rise of standardisation and certification?. Computer Law & Security Review. 2025 Apr 1;56:106093.